NASA-CR-191,466

NASA Contractor Report 191466

NASA-CR-191466
19940017941

Towards the Formal Verification of the Requirements and Design of a Processor Interface Unit — HOL Listings

David A. Fura
The Boeing Company
Seattle, Washington

Phillip J. Windley
University of Idaho
Moscow, Idaho

Gerald C. Cohen
The Boeing Company
Seattle, Washington

NASA Contract NAS1-18586
November 1993

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-0001

Preface

This document was generated in support of NASA contract NAS1-18586, Design and Validation of Digital Flight Control Systems Suitable for Fly-By-Wire Applications, Task Assignment 10. Task 10 is concerned with the formal specification and verification of a processor interface unit.

This report contains the HOL listings of the formal verification of the design and partial requirements for a processor interface unit using the HOL theorem-proving system. The verification approach is described in NASA CR-4522. The processor interface unit is a single-chip subsystem within a fault-tolerant embedded system under development within the Boeing Defense & Space Group. It provides the opportunity to investigate the specification and verification of a real-world subsystem within a commercially-developed fault-tolerant computer.

The NASA technical monitor for this work is Sally Johnson of the NASA Langley Research Center, Hampton, Virginia.

The work was accomplished at the Boeing Company, Seattle, Washington and the University of Idaho, Moscow, Idaho. Personnel responsible for the work include:

Boeing Defense & Space Group:
D. Gangsaas, Responsible Manager
T. M. Richardson, Program Manager

Boeing Defense & Space Group:
Gerald C. Cohen, Principal Investigator
David A. Fura, Researcher

University of Idaho:
Dr. Phillip J. Windley, Chief Researcher

Contents

1 Introduction

This technical report contains the HOL listings of the partial verification of the requirements and design for a commercially-developed processor interface unit (or PIU). The PIU is an interface chip performing memory-interface, bus-interface, and additional support services for a commercial microprocessor within a fault-tolerant computer system. This system, the Fault-Tolerant Embedded Processor (FTEP), is targeted towards applications in avionics and space requiring extremely high levels of mission reliability, extended maintenance-free operation, or both.

This report contains the actual HOL listings of the PIU verification as it currently exists. For those interested in an informal description of the PIU verification, NASA CR-4522 contains a discussion of the issues involved in the PIU verification, as well as an overview of the verification itself.

Section 2 of this report contains general-purpose HOL theories and definitions that support the PIU verification. These include arithmetic theories dealing with inequalities and associativity, and a collection of tactics used in the PIU proofs.

Section 3 contains the HOL listings for the completed PIU *design* verification.

Section 4 contains the HOL listings for the partial *requirements* verification of the P-Port.

2 Supporting Theories

This section contains general-purpose theories and loadable definitions used in the PIU verification. The file *aux_defs.ml* contains some useful definitions and tactics; the file *abs_theory.ml* contains the definitions for creating abstract theories in HOL; and the file *pt_defs.ml* contains some definitions used in the PIU verification. The files *pt_tacs.ml* and *range_induct.ml* contains assumption-list-handling tactics and a range-induction tactic, respectively.

The three theories *assoc*, *cond*, and *ineq* contain theorems involving arithmetic associativity, the HOL conditional operator, and arithmetic inequalities, respectively.

```
%----------------------------------------------------------------

   File: aux_defs.ml

   Author: PJWindley (UCDavis)

   Purpose: Gives a few useful definitions.

----------------------------------------------------------------%

let new_autoload_theory thy =
 map (\name. autoload_theory('definition',thy,name))
     (map fst (definitions thy));
 map (\name. autoload_theory('theorem',thy,name))
     (map fst (theorems thy));
 ();;

let load_parent s =
   new_parent s;
   new_autoload_theory s;;

let toggle_flag s =
   set_flag(s,not(get_flag_value s));;

% flip an equailty thm %
let SYM_RULE =
  (CONV_RULE (ONCE_DEPTH_CONV SYM_CONV))
? failwith 'SYM_RULE';;

% expand let definitions %
% changed "ONCE_" to "PURE_ONCE_" 28Sep92 [DAF] %
let EXPAND_LET_RULE x =
   (BETA_RULE
      (PURE_ONCE_REWRITE_RULE [LET_DEF] x));;

let EXPAND_LET_TAC =
   ONCE_REWRITE_TAC [LET_DEF]
   THEN BETA_TAC;;

% instantiate types before specing %
let ISPEC var thm =
      (let _,i = match (fst(dest_forall (concl thm))) var in
       SPEC var (INST_TYPE i thm)) ? failwith 'ISPEC';;

% beta reduce a pair %
let PAIR_BETA_RULE =
    BETA_RULE o (ONCE_REWRITE_RULE [UNCURRY_DEF]);;

let PAIR_BETA_TAC =
    ONCE_REWRITE_TAC [UNCURRY_DEF] THEN
    BETA_TAC ;;


%----------------------------------------------------------------
```

```
Redefine TAC_PROOF so that it tells you about the unsolved goals.
-----------------------------------------------------------------%
let TAC_PROOF : (goal # tactic) -> thm =
    set_fail_prefix 'TAC_PROOF'
       (\(g,tac).
           let gl,p = tac g in
           if null gl then p[]
           else (
              message ('Unsolved goals:');
              map print_goal gl;
              print_newline();
              failwith 'unsolved goals'));;



%-----------------------------------------------------------------
 Removed from HOL 1.12.
-----------------------------------------------------------------%
let int_to_term = ((C o curry) mk_const ":num") o string_of_int and
    term_to_int = (int_of_string o fst o dest_const);;


%-----------------------------------------------------------------

   File:         abs_theory.ml

   Description:

   Defines ML functions for defining generic structures.  This is
   a refinement of the abstract.ml package.

   Author:       (c) P. J. Windley 1991

   Date:         07 NOV 91

   Modification History:

   07NOV91 --- [PJW] Original file.

   22JAN92 --- [PJW] Changed make_abs_goal to account for different
               type instantiations.  This allows proof of goals about
               more than one instance of a generic object.

   07JUN92 --- [PJW] Added sections to hide internal functions.
               Changed new_theory_obligation to take only one
               thob instead of a list of thobs.
               Added definition of STRIP_THOBS_THEN and redefined
               STRIP_THOBS_TAC in terms of it.
               Changed implementation of instantiate_abstract_theorems
               to allow the use of explicit theory obligations.

-----------------------------------------------------------------%

%-----------------------------------------------------------------

   To Do:

   Extending absract representations---
       This is difficult since HOL doesn't handle subtypes.

-----------------------------------------------------------------%

print_newline();;
message('loading abs_theory');;

%-----------------------------------------------------------------
 Extend the help search path
-----------------------------------------------------------------%
tty_write 'Extending help search path';
let path = library_pathname()^'/abs_theory/help/entries/' in
    set_help_search_path (union [path] (help_search_path()));;
```

```
begin_section abs_theory;;

%-----------------------------------------------------------------

 load the patch to concat if the following ml statement prints a
 number less than 5000 and HOL version <= 2.00.

 lisp '(print call-arguments-limit)';;


let test_string n =
    letrec aux s i =
        (i = 0) => s | aux ('x'^s) (i-1) in
    aux '' n;;

-----------------------------------------------------------------%

% message 'patching concat';loadf 'concat-patch';; %

%-----------------------------------------------------------------
 define some general list functions
-----------------------------------------------------------------%

let int_to_term = (((C (curry mk_const)) ":num") o string_of_int) and
    term_to_int = (int_of_string o fst o dest_const);;

letrec for n lst =
    (n=0) => []
           | (hd lst).(for (n-1) (tl lst));;

% function composition over a list of functions %
letrec ol lst = null(lst) => I | (hd lst) o (ol (tl lst)) ;;

%-----------------------------------------------------------------
 General purpose inference rules
-----------------------------------------------------------------%
let X_SPEC tm v thm =
    let tml = (fst o strip_forall o concl) thm in
    letrec SPEC1_aux tml sthm =
        (tml = []) => sthm |
        ((hd tml) = tm) =>
           (SPEC v sthm) |
           (GEN (hd tml) (SPEC1_aux (tl tml) (SPEC (hd tml) sthm))) in
    SPEC1_aux tml thm;;

letrec CONJ_IMP th =
    let w = concl th in
    if is_imp w then
        let ante,conc = dest_imp w in
        if is_conj ante then
            let a,b = dest_conj ante in
            CONJ_IMP
            (DISCH a (DISCH b (MP th (CONJ (ASSUME a) (ASSUME b)))))
        else (DISCH ante (CONJ_IMP (UNDISCH th)))
   else th;;

%-----------------------------------------------------------------
 define auxilliary functions for new_abstract_representation
-----------------------------------------------------------------%

let abs_type_info = (type_of o snd o dest_comb o fst o dest_eq o snd o
                     strip_forall o snd o dest_abs o snd o dest_comb o
                     snd o strip_forall o concl) ;;

let dest_all_type ty =
     is_vartype ty => (dest_vartype ty,[]:(type)list) | dest_type ty;;

let string_from_type ty =
    letrec string_aux ty =
let s,tl = dest_all_type ty in
null tl => s |
let sl = map string_aux tl in
'('^(itlist (\x y. x^','^y) (butlast sl) (last sl))^')'^s in
    (string_aux ty)^' ';;


let ty_str name lst =
      name^' = '^name^' '^
        (itlist (\x y. x^'  '^y) (map string_from_type lst) ''));;

letref def_prefix = 'abs_def_';;

%---------------------------------------------------------------
 new_abstract_representation:

 Defines type for representation and selectors on type.
---------------------------------------------------------------%

let new_abstract_representation name lst =
    let nl,tl = split lst in
    let ty_axiom = define_type name (ty_str name tl) in
    let cons_term =
       (snd o dest_comb o fst o dest_eq o snd o strip_forall o snd o
        dest_abs o snd o dest_comb o snd o strip_forall o concl) ty_axiom in
    let make_rec_def (x, y) =
new_recursive_definition false ty_axiom (def_prefix^x)
"^(mk_var (x,mk_type('fun',[type_of cons_term;
      type_of y]))) ^cons_term = ^y" in
     (map2 make_rec_def (nl,(snd o strip_comb) cons_term));ty_axiom;;


let get_abs_defs theory =
    let prefix_len = length o explode in
    map snd (
    mapfilter (\x. (((for (prefix_len def_prefix)) o explode o fst) x) =
                (explode def_prefix) => x | fail)
              (definitions theory));;


let instantiate_abstract_definition th_name def_name defn2 inst_list =
    let def = definition th_name def_name in
    let inst_def =
        GEN_ALL (
        BETA_RULE (
        REWRITE_RULE (get_abs_defs th_name) (
        o1 (map (\(x,y).INST_TY_TERM (match x y)) inst_list)
                 (SPEC_ALL def)))) in
    let new_def =
        GEN_ALL (
        BETA_RULE (
        REWRITE_RULE (get_abs_defs th_name) (
        o1 (map (\(x,y).INST_TY_TERM (match x y)) inst_list)
                 (SPEC_ALL defn2)))) in
    BETA_RULE (ONCE_REWRITE_RULE [inst_def] new_def);;


%---------------------------------------------------------------
 Auxilliary definitions for theory obligations
---------------------------------------------------------------%

letref thobs = []:(type#thm)list;;

let thobs_prefix = 'thobs_';;

let new_theory_obligations stm_pair =
    let get_thob_type thm =
 ((type_of o rand o fst o dest_eq o
          snd o strip_forall o concl) thm) in
    let is_not_pred_def tm =
        not((type_of o snd o dest_eq o
             snd o strip_forall o snd) tm = ":bool") in
%    let not_bool_list = (filter is_not_pred_def stm_pair) in %
    let make_def (x,y) = (
        let new_def = new_definition (thobs_prefix^x, y) in
        (get_thob_type new_def,new_def)) in
    (is_not_pred_def stm_pair) =>
        failwith 'NON PREDICATE TERMS IN THEORY OBLIGATIONS' |
        (thobs := [make_def stm_pair] @ thobs);();;


let get_thobs theory =
    let prefix_len = length o explode in
    let get_thob_type thm =
 ((type_of o rand o fst o dest_eq o
          snd o strip_forall o concl) thm) in
    let the_thobs =
        filter (\x. (((for (prefix_len thobs_prefix)) o explode o fst) x) =
       (explode thobs_prefix))
 (definitions theory) in
    thobs := (map (\(x,y). (get_thob_type y, y)) the_thobs) @ thobs;
    thobs;;


%-------------------------------------------------------------------
 Functions for proving abstract goals.
-------------------------------------------------------------------%

let orelsef f g x = (f x) ? (g x);;

ml_curried_infix 'orelsef';;

let D (f,g) x = (f x,g x);;

ml_paired_infix 'D';;

let make_abs_goal (hyps,go) =
    if null(thobs) then failwith 'No theory obligations defined.' else
    let vl,pred = strip_forall go in
    let ql = union vl (frees pred) in
    let type_cons_of = fst o (dest_type orelsef (dest_vartype D \x.[])) in
    let thob_types = map (type_cons_of o fst) thobs in
    let tmp_goal = list_mk_forall
  (filter (\x. not(mem ((type_cons_of o type_of) x) thob_types)) ql,
   pred) in
    let vars = filter (\x. mem ((type_cons_of o type_of) x) thob_types) ql in
    let make_hyps var =
let get_thob_var tm =
    ((rand o fst o dest_eq o snd o strip_forall ) tm) in
let thob = snd (((assoc o type_cons_of o type_of) var)
                         (map (type_cons_of # I) thobs)) in
(conjuncts o snd o dest_eq o concl o
 (INST_TY_TERM (match ((get_thob_var o concl) thob)
                                             var)) o
 SPEC_ALL) thob in
    (hyps@(flat (map make_hyps vars)), tmp_goal);;

%Prove and store an abstract theorem%
let prove_abs_thm(tok, w, tac:tactic) =
    let gl,prf = tac (make_abs_goal ([],w)) in
    if null gl then save_thm (tok, prf[])
    else
       (message ('Unsolved goals:');
        map print_goal gl;
        print_newline();
        failwith ('prove_thm -- could not prove ' ^ tok));;


% ABS_TAC_PROOF (g,tac) uses tac to prove the abstract goal g      %
let ABS_TAC_PROOF : (goal # tactic) -> thm =
    set_fail_prefix 'ABS_TAC_PROOF'
       (\(g,tac).
           let new_g = (make_abs_goal g) in
           let gl,p = tac new_g in
           if null gl then p[]
           else (
              message ('Unsolved goals:');
              map print_goal gl;
              print_newline();
              failwith 'unsolved goals'));;


%Set the top-level goal, initialize %
let set_abs_goal g =
           let new_g = (make_abs_goal g) in
           change_state (abs_goals (new_stack new_g));;

let g = \t. set_abs_goal([],t);;

let STRIP_THOBS_THEN ttac =
    if null(thobs) then failwith 'No theory obligations defined.' else (
    REPEAT GEN_TAC THEN
    STRIP_GOAL_THEN
      (ttac o (REWRITE_RULE (map snd thobs))));;


let STRIP_THOBS_TAC ((asl,thm):goal) =
     STRIP_THOBS_THEN STRIP_ASSUME_TAC(asl,thm);;


%-----------------------------------------------------------------
  Functions for USING an abstract theory
-----------------------------------------------------------------%

let new_abstract_parent s =
    new_parent s;
    get_thobs s;
    ();;

let EXPAND_THOBS_TAC name =
    REWRITE_TAC ((map snd thobs) @ get_abs_defs name);;


let instantiate_abstract_theorem th_name thm_name inst_list lemma_list =
    let thm =  (theorem th_name thm_name) in
    let inst_thm =
        CONJ_IMP (
BETA_RULE (
REWRITE_RULE ((map snd thobs) @ get_abs_defs th_name) (
(ol (map (\(x,y).INST_TY_TERM (match x y)) inst_list))(
(DISCH_ALL o (ol (map (\l.(X_SPEC (fst l) (fst l))) inst_list)))
         thm)))) in
    let mk_thm_list =
         CONJUNCTS o BETA_RULE o
          (REWRITE_RULE ((map snd thobs) @ get_abs_defs th_name)) in
    let thm_list =
map (\y. find (\x.(concl x) = y)
 (flat (map mk_thm_list lemma_list)))
    ((hyp o UNDISCH_ALL o CONJ_IMP) inst_thm) in
    LIST_MP thm_list inst_thm;;

%-----------------------------------------------------------------
 Modify the standard commands so that they know about obligation
 lists.
-----------------------------------------------------------------%
let close_theory_orig = close_theory;;

let close_theory x =
   thobs := [];
   close_theory_orig x;;

let new_theory_orig = new_theory;;

let new_theory x =
   thobs := [];
   new_theory_orig x;;

%-----------------------------------------------------------------
```

```
 Bind exportable functions to it
-----------------------------------------------------------------%
(
ABS_TAC_PROOF,
EXPAND_THOBS_TAC,
STRIP_THOBS_TAC,
STRIP_THOBS_THEN,
abs_type_info,
close_theory,
g,
instantiate_abstract_definition,
instantiate_abstract_theorem,
new_abstract_parent,
new_abstract_representation,
new_theory,
new_theory_obligations,
prove_abs_thm,
set_abs_goal
);;

end_section abs_theory;;

%-----------------------------------------------------------------
 Recover exportable functions from it.
-----------------------------------------------------------------%
let (
ABS_TAC_PROOF,
EXPAND_THOBS_TAC,
STRIP_THOBS_TAC,
STRIP_THOBS_THEN,
abs_type_info,
close_theory,
g,
instantiate_abstract_definition,
instantiate_abstract_theorem,
new_abstract_parent,
new_abstract_representation,
new_theory,
new_theory_obligations,
prove_abs_thm,
set_abs_goal
) = it;;


%--------------------------------------------------------------------------------

   File:     pt_defs.ml

   Author:   (c) D.A. Fura 1992-93

   Date:     15 February 1993

   Definitions used in the P-Port trans-level proof.

--------------------------------------------------------------------------------%

let New_State_Is_PA = new_definition
   ('New_State_Is_PA',
    "! (s' :timeC->pc_state) (e' :timeC->pc_env) (t' :timeC) .
     New_State_Is_PA s' e' t' =
        ((P_fsm_rstS(s' t') => PA |
        ((P_fsm_stateS(s' t') = PH) =>
         (P_fsm_hold_S(s' t') => PA | PH) |
         ((P_fsm_stateS(s' t') = PA) =>
          ((P_fsm_mrqtS(s' t') \/
            ~P_fsm_crqt_S(s' t') /\ ~P_fsm_cgnt_S(s' t')) => PD |
           ((~P_fsm_hold_S(s' t') /\ P_fsm_lock_S(s' t')) => PH | PA)) |
          ((P_fsm_sackS(s' t') /\ P_fsm_hold_S(s' t')) => PA |
           ((P_fsm_sackS(s' t') /\
             ~P_fsm_hold_S(s' t') /\
             ~P_fsm_lock_S(s' t')) => PA |
            ((P_fsm_sackS(s' t') /\
                ~P_fsm_hold_S(s' t') /\
                P_fsm_lock_S(s' t')) => PH | PD)))))) = PA)"
   );;

let New_State_Is_PD = new_definition
   ('New_State_Is_PD',
    "! (s' :timeC->pc_state) (e' :timeC->pc_env) (t' :timeC) .
     New_State_Is_PD s' e' t' =
        ((P_fsm_rstS(s' t') => PA |
        ((P_fsm_stateS(s' t') = PH) =>
         (P_fsm_hold_S(s' t') => PA | PH) |
         ((P_fsm_stateS(s' t') = PA) =>
          ((P_fsm_mrqtS(s' t') \/
            ~P_fsm_crqt_S(s' t') /\ ~P_fsm_cgnt_S(s' t')) => PD |
           ((~P_fsm_hold_S(s' t') /\ P_fsm_lock_S(s' t')) => PH | PA)) |
          ((P_fsm_sackS(s' t') /\ P_fsm_hold_S(s' t')) => PA |
           ((P_fsm_sackS(s' t') /\
             ~P_fsm_hold_S(s' t') /\
             ~P_fsm_lock_S(s' t')) => PA |
            ((P_fsm_sackS(s' t') /\
              ~P_fsm_hold_S(s' t') /\
              P_fsm_lock_S(s' t')) => PH | PD)))))) = PD)"
   );;

let New_State_Is_PH = new_definition
   ('New_State_Is_PH',
    "! (s' :timeC->pc_state) (e' :timeC->pc_env) (t' :timeC) .
     New_State_Is_PH s' e' t' =
        ((P_fsm_rstS(s' t') => PA |
        ((P_fsm_stateS(s' t') = PH) =>
         (P_fsm_hold_S(s' t') => PA | PH) |
         ((P_fsm_stateS(s' t') = PA) =>
          ((P_fsm_mrqtS(s' t') \/
            ~P_fsm_crqt_S(s' t') /\ ~P_fsm_cgnt_S(s' t')) => PD |
           ((~P_fsm_hold_S(s' t') /\ P_fsm_lock_S(s' t')) => PH | PA)) |
          ((P_fsm_sackS(s' t') /\ P_fsm_hold_S(s' t')) => PA |
           ((P_fsm_sackS(s' t') /\
             ~P_fsm_hold_S(s' t') /\
             ~P_fsm_lock_S(s' t')) => PA |
            ((P_fsm_sackS(s' t') /\
              ~P_fsm_hold_S(s' t') /\
              P_fsm_lock_S(s' t')) => PH | PD)))))) = PH)"
   );;

let New_P_Rqt_Is_TRUE = new_definition
   ('New_P_Rqt_Is_TRUE',
    "! (s' :timeC->pc_state) (e' :timeC->pc_env) (t' :timeC) .
     New_P_Rqt_Is_TRUE s' e' t' =
        (~SND(L_ads_E(e' t')) /\ SND(L_den_E(e' t')) \/
         SND(RstE(e' t')) \/
         (P_sizeS(s' t') = (P_downS(s' t') => WORDN 1 1 | WORDN 1 0)) /\
         ~SND(I_srdy_E(e' t')) /\
         New_State_Is_PD s' e' t') =>
        (((~SND(L_ads_E(e' t')) /\ SND(L_den_E(e' t'))) /\
          ~(SND(RstE(e' t')) \/
            (P_sizeS(s' t') = (P_downS(s' t') => WORDN 1 1 | WORDN 1 0)) /\
            ~SND(I_srdy_E(e' t')) /\
            New_State_Is_PD s' e' t')) => T |
         ((~(~SND(L_ads_E(e' t')) /\ SND(L_den_E(e' t'))) /\
           (SND(RstE(e' t')) \/
            (P_sizeS(s' t') = (P_downS(s' t') => WORDN 1 1 | WORDN 1 0)) /\
            ~SND(I_srdy_E(e' t')) /\
            New_State_Is_PD s' e' t')) => F |
          ((~(~SND(L_ads_E(e' t')) /\ SND(L_den_E(e' t'))) /\
            ~(SND(RstE(e' t')) \/
              (P_sizeS(s' t') = (P_downS(s' t') => WORDN 1 1 | WORDN 1 0)) /\
              ~SND(I_srdy_E(e' t')) /\
              New_State_Is_PD s' e' t')) => F | ARB))) | P_rqtS(s' t')"
   );;

let Sack_Sig_Is_TRUE = new_definition
   ('Sack_Sig_Is_TRUE',
    "! (s :timeC->pc_state) (e :timeC->pc_env) .
     Sack_Sig_Is_TRUE s e =
       (\u:timeC.
         (P_sizeS(s u) = (P_downS(s u) => WORDN 1 1 | WORDN 1 0)) /\
         ~SND(I_srdy_E(e u)) /\
         New_State_Is_PD s e u)"
   );;

let P_Size_Zero_Out_Is_TRUE = new_definition
   ('P_Size_Zero_Out_Is_TRUE',
    "! (s' :timeC->pc_state) (t' :timeC) .
     P_Size_Zero_Out_Is_TRUE s' t' =
        P_sizeS(s' t') = (P_downS(s' t') => WORDN 1 1 | WORDN 1 0)"
   );;


%-------------------------------------------------------------------------------

   File:     pt_tacs.ml

   Author:   (c) D.A. Fura 1992-93

   Date:     16 February 1993

   Custom tactics used in the P-Port trans-level proof.

-------------------------------------------------------------------------------%

let LIMP = (\thm. GEN_ALL (fst (EQ_IMP_RULE (SPEC_ALL thm))));;
let RIMP = (\thm. GEN_ALL (snd (EQ_IMP_RULE (SPEC_ALL thm))));;

%--------------------------------------------------------------

   Code from Brian Graham.  See "Dealing with the Choice Operator
   in HOL88" by Brian Graham for more information.

--------------------------------------------------------------%

%
   SELECT_UNIQUE_RULE:
   ===================

   ("x","y")   A1 |- Q[y]  A2 |- !x y.(Q[x]/\Q[y]) ==> (x=y)
   =========================================================
      A1 U A2 |- (@x.Q[x]) = y

Permits substitution for values specified by the Hilbert Choice
operator with a specific value, if and only if unique existance
of the specific value is proven.
%

let in1_conv_rule = CONV_RULE o ONCE_DEPTH_CONV o CHANGED_CONV;;

let SELECT_UNIQUE_RULE (x,y) th1 th2 =
  let Q = mk_abs (x, subst [x,y] (concl th1))
  in
  let th1' = SUBST [SYM (BETA_CONV "^Q ^y"), "b:bool"] "b:bool" th1
  in
  (MP (SPECL ["$@ ^Q"; y] th2)
      (CONJ (in1_conv_rule BETA_CONV (SELECT_INTRO th1')) th1));;

%
   SELECT_UNIQUE_TAC:
   ==================

            [ A ] "(@x. Q[x]) = y"
   ==================================================
        [ A ] "Q[y]"   [ A ] "!x y.(Q[x]/\Q[y]) ==> (x=y)"

Given a goal that requires proof of the value specified by the
Hilbert choice operator, it returns 2 subgoals:
  1. "y" satisfies the predicate, and
  2. unique existance of the value that satisfies the predicate.
%

let SELECT_UNIQUE_TAC:tactic (gl,g) =
  let Q,y = dest_eq g
  in
  let x,Qx = dest_select Q
  in
  let x' = variant (x.freesl(g.gl))x
  in
  let Qx' = subst [x', x] Qx
  in
  ([gl,subst [y,x]Qx;
    gl, "!^x ^x'. (^Qx /\ ^Qx') ==> (^x = ^x')"],
   (\thl. SELECT_UNIQUE_RULE (x,y) (hd thl) (hd (tl thl))));;

letrec Find_Imp tm =
   let ret_pair =
      if is_imp tm & is_exists(fst(dest_imp tm)) then ('exl',tm)
      else if is_imp tm & is_exists(snd(dest_imp tm)) then ('exr',tm)
      else if is_imp tm & is_forall(fst(dest_imp tm)) then ('forl',tm)
      else if is_imp tm & is_forall(snd(dest_imp tm)) then ('forr',tm)
      else if is_imp tm then
              if not(fst(Find_Imp(fst(dest_imp tm))) = 'n') then
                 Find_Imp(fst(dest_imp tm))
              else Find_Imp(snd(dest_imp tm))
      else if is_forall tm then Find_Imp(snd(dest_forall tm))
      else if is_exists tm then Find_Imp(snd(dest_exists tm))
      else ('n',tm)
   in ret_pair;;

let QUANT_OUT_IMP_TAC :tactic =
   \(asl,w).
      let str,tm = Find_Imp w in
      if str = 'forl' then REWRITE_TAC [LEFT_IMP_FORALL_CONV tm] (asl,w)
      else if str = 'forr' then REWRITE_TAC [RIGHT_IMP_FORALL_CONV tm] (asl,w)
      else if str = 'exl' then REWRITE_TAC [LEFT_IMP_EXISTS_CONV tm] (asl,w)
      else if str = 'exr' then REWRITE_TAC [RIGHT_IMP_EXISTS_CONV tm] (asl,w)
      else NO_TAC (asl,w)
      ? failwith 'QUANT_OUT_IMP_TAC';;

letrec Find_Asm_Tm (w,tm) =
   if is_eq w & (rhs w = tm) then ('y',w)
   else
     if is_imp w
       then if fst(Find_Asm_Tm(fst(dest_imp w),tm)) = 'y'
              then Find_Asm_Tm(fst(dest_imp w),tm)
            else if fst(Find_Asm_Tm(snd(dest_imp w),tm)) = 'y'
                   then Find_Asm_Tm(snd(dest_imp w),tm)
                 else ('n',w)
     else if is_neg w
            then if fst(Find_Asm_Tm((dest_neg w),tm)) = 'y'
                   then Find_Asm_Tm((dest_neg w),tm)
                 else ('n',w)
     else if is_cond w
            then if fst(Find_Asm_Tm(fst(dest_cond w),tm)) = 'y'
                   then Find_Asm_Tm(fst(dest_cond w),tm)
            else if fst(Find_Asm_Tm(fst(snd(dest_cond w)),tm)) = 'y'
                   then Find_Asm_Tm(fst(snd(dest_cond w)),tm)
            else if fst(Find_Asm_Tm(snd(snd(dest_cond w)),tm)) = 'y'
                   then Find_Asm_Tm(snd(snd(dest_cond w)),tm)
                 else ('n',w)
     else if is_eq w
            then if fst(Find_Asm_Tm(fst(dest_eq w),tm)) = 'y'
                   then Find_Asm_Tm(fst(dest_eq w),tm)
            else if fst(Find_Asm_Tm(snd(dest_eq w),tm)) = 'y'
                   then Find_Asm_Tm(snd(dest_eq w),tm)
                 else ('n',w)
     else ('n',w);;

let ASM_CASES_MATCH_RHS_TAC tm :tactic =
   \(asl,w).
```

```
      let str,tm_ret = Find_Asm_Tm (w,tm) in
      if str = 'y' then ASM_CASES_TAC tm_ret (asl,w)
      else ALL_TAC (asl,w)
      ? failwith 'ASM_CASES_MATCH_RHS_TAC';;

let NO_IMP_ASSUME_TAC thm :tactic =
   \(asl,w).
      let stripped_thm = snd(strip_forall(snd(dest_thm thm))) in
      if is_imp(stripped_thm) & not(snd(dest_imp(stripped_thm)) = "F")
         then ALL_TAC (asl,w)
      else ASSUME_TAC thm (asl,w)
      ? failwith 'NO_IMP_ASSUME_TAC';;

let NO_UNQUANT_IMP_ASSUME_TAC thm :tactic =
   \(asl,w).
      let stripped_thm = snd(dest_thm thm) in
      if is_imp(stripped_thm) & not(snd(dest_imp(stripped_thm)) = "F")
         then ALL_TAC (asl,w)
      else ASSUME_TAC thm (asl,w)
      ? failwith 'NO_IMP_ASSUME_TAC';;

let NO_TRUTH_ASSUME_TAC thm :tactic =
   \(asl,w).
      if snd(dest_thm thm) = "T" then ALL_TAC (asl,w)
      else ASSUME_TAC thm (asl,w)
      ? failwith 'NO_TRUTH_ASSUME_TAC';;

let SPEC_UNDISCH_TAC (tm1,tm2) :tactic =
   \(asl,w).
      (UNDISCH_TAC tm1
       THEN PURE_ONCE_REWRITE_TAC [LEFT_IMP_FORALL_CONV (mk_imp (tm1,w))]
       THEN EXISTS_TAC tm2) (asl,w)
       ? failwith 'SPEC_UNDISCH_TAC';;

let NRULE_ASSUM_TAC (tm,rul) :tactic =
   \(asl,w).
      let f =
         (\thm.
             if snd(dest_thm thm) = tm then ASSUME_TAC (rul thm)
             else ASSUME_TAC thm) in
      POP_ASSUM_LIST (MAP_EVERY (\thm. f thm)) ((rev asl),w)
      ? failwith 'NRULE_ASSUM_TAC';;

let SPEC_ASSUM_TAC (tm1,tm2) :tactic =
   \(asl,w).
      let f =
        (\thm.
           if snd(dest_thm thm) = tm1 then ASSUME_TAC (SPEC tm2 thm)
           else ASSUME_TAC thm) in
      POP_ASSUM_LIST (MAP_EVERY (\thm. f thm)) ((rev asl),w)
      ? failwith 'SPEC_ASSUM_TAC';;

let SPECL_ASSUM_TAC (tm1,tm2list) :tactic =
   \(asl,w).
      let f =
        (\thm.
           if snd(dest_thm thm) = tm1 then ASSUME_TAC (SPECL tm2list thm)
           else ASSUME_TAC thm) in
      POP_ASSUM_LIST (MAP_EVERY (\thm. f thm)) ((rev asl),w)
      ? failwith 'SPEC_ASSUM_TAC';;

let GEN_ASSUM_TAC (tm1,tm2) :tactic =
   \(asl,w).
      let f =
        (\thm.
           if snd(dest_thm thm) = tm1 then ASSUME_TAC (GEN tm2 thm)
           else ASSUME_TAC thm) in
      POP_ASSUM_LIST (MAP_EVERY (\thm. f thm)) ((rev asl),w)
      ? failwith 'GEN_ASSUM_TAC';;

let CHOOSE_ASSUM_TAC tm1 :tactic =
   \(asl,w).
```

```
        (UNDISCH_TAC tm1
         THEN PURE_ONCE_REWRITE_TAC [LEFT_IMP_EXISTS_CONV (mk_imp (tm1,w))]
         THEN GEN_TAC
         THEN DISCH_TAC) (asl,w)
         ? failwith 'CHOOSE_ASSUM_TAC';;

let REWRITE_ASSUM_TAC (tm,tlist) :tactic =
   \(asl,w).
      let f =
        (\thm.
           if snd(dest_thm thm) = tm then ASSUME_TAC (REWRITE_RULE tlist thm)
           else ASSUME_TAC thm) in
      POP_ASSUM_LIST (MAP_EVERY (\thm. f thm)) ((rev asl),w)
      ? failwith 'REWRITE_ASSUM_TAC';;

let PURE_REWRITE_ASSUM_TAC (tm,tlist) :tactic =
   \(asl,w).
      let f =
         (\thm.
             if snd(dest_thm thm) = tm
                 then ASSUME_TAC (PURE_REWRITE_RULE tlist thm)
             else ASSUME_TAC thm) in
      POP_ASSUM_LIST (MAP_EVERY (\thm. f thm)) ((rev asl),w)
      ? failwith 'PURE_REWRITE_ASSUM_TAC';;

let PURE_REWRITE_ASSUM_TAC (tm,tlist) :tactic =
   \(asl,w).
      (UNDISCH_TAC tm
       THEN PURE_REWRITE_TAC tlist
       THEN DISCH_TAC) (asl,w)
       ? failwith 'PURE_REWRITE_ASSUM_TAC';;

let PURE_ONCE_REWRITE_ASSUM_TAC (tm,tlist) :tactic =
   \(asl,w).
      let f =
         (\thm.
             if snd(dest_thm thm) = tm
                 then ASSUME_TAC (PURE_ONCE_REWRITE_RULE tlist thm)
             else ASSUME_TAC thm) in
      POP_ASSUM_LIST (MAP_EVERY (\thm. f thm)) ((rev asl),w)
      ? failwith 'PURE_ONCE_REWRITE_ASSUM_TAC';;

let DELETE_ASSUM_TAC tm :tactic =
   \(asl,w).
      let f =
         (\thm.
             if snd(dest_thm thm) = tm
                 then ALL_TAC
             else ASSUME_TAC thm) in
      POP_ASSUM_LIST (MAP_EVERY (\thm. f thm)) ((rev asl),w)
      ? failwith 'PURE_ONCE_REWRITE_ASSUM_TAC';;

let ASM_REWRITE_ASSUM_TAC (tm,tlist) :tactic =
   \(asl,w).
      (UNDISCH_TAC tm
       THEN ASM_REWRITE_TAC tlist
       THEN DISCH_TAC) (asl,w)
       ? failwith 'ASM_REWRITE_ASSUM_TAC';;

let REWRITE_SPEC_ASSUM_TAC (tm1,tm2,tlist) :tactic =
   \(asl,w).
      (UNDISCH_TAC tm1
       THEN PURE_ONCE_REWRITE_TAC [LEFT_IMP_FORALL_CONV (mk_imp (tm1,w))]
       THEN EXISTS_TAC tm2
       THEN REWRITE_TAC tlist
       THEN DISCH_TAC) (asl,w)
       ? failwith 'REWRITE_SPEC_ASSUM_TAC';;

let ASM_REWRITE_SPEC_ASSUM_TAC (tm1,tm2,tlist) :tactic =
   \(asl,w).
      (UNDISCH_TAC tm1
       THEN PURE_ONCE_REWRITE_TAC [LEFT_IMP_FORALL_CONV (mk_imp (tm1,w))]
       THEN EXISTS_TAC tm2
       THEN ASM_REWRITE_TAC tlist
       THEN DISCH_TAC) (asl,w)
       ? failwith 'ASM_REWRITE_SPEC_ASSUM_TAC';;

let USTRIP_REWRITE_ASSUM_TAC (tm,tlist) :tactic =
   \(asl,w).
      (UNDISCH_TAC tm
       THEN REWRITE_TAC tlist
       THEN BETA_TAC
       THEN REPEAT QUANT_OUT_IMP_TAC
       THEN REDUCE_TAC
       THEN REPEAT STRIP_TAC) (asl,w)
       ? failwith 'USTRIP_REWRITE_ASSUM_TAC';;

let USTRIP_ASM_REWRITE_ASSUM_TAC (tm,tlist) :tactic =
   \(asl,w).
      (UNDISCH_TAC tm
       THEN ASM_REWRITE_TAC tlist
       THEN BETA_TAC
       THEN REPEAT QUANT_OUT_IMP_TAC
       THEN REDUCE_TAC
       THEN REPEAT STRIP_TAC) (asl,w)
       ? failwith 'USTRIP_ASM_REWRITE_ASSUM_TAC';;

let UNDISCH_MATCH_LHS_TAC tm :tactic =
   \(asl,w).
      let f =
         (\thm.
            let stripped_thm = snd(strip_forall(snd(dest_thm thm))) in
            if is_eq(stripped_thm)
               then if lhs(stripped_thm) = tm
                       then UNDISCH_TAC (snd(dest_thm thm))
                    else ALL_TAC
            else ALL_TAC) in
      ASSUM_LIST (MAP_EVERY (\thm. f thm)) (asl,w)
      ? failwith 'UNDISCH_MATCH_LHS_TAC';;

let SPEC_UNDISCH_MATCH_LHS_TAC (tm1,tm2) :tactic =
   \(asl,w).
      let f =
         (\thm.
            let stripped_thm = snd(strip_forall(snd(dest_thm thm))) in
            if is_eq(stripped_thm)
               then if lhs(stripped_thm) = tm1
                       then (UNDISCH_TAC (snd(dest_thm thm))
                             THEN PURE_ONCE_REWRITE_TAC
                                     [LEFT_IMP_FORALL_CONV
                                        (mk_imp (snd(dest_thm thm),w))]
                             THEN EXISTS_TAC tm2)
                    else ALL_TAC
            else ALL_TAC) in
      ASSUM_LIST (MAP_EVERY (\thm. f thm)) (asl,w)
      ? failwith 'SPEC_UNDISCH_MATCH_LHS_TAC';;

let SPEC_UNDISCH_MATCH_RHS_TAC (tm1,tm2) :tactic =
   \(asl,w).
      let f =
         (\thm.
            let stripped_thm = snd(strip_forall(snd(dest_thm thm))) in
            if is_eq(stripped_thm)
               then if rhs(stripped_thm) = tm1
                       then (UNDISCH_TAC (snd(dest_thm thm))
                             THEN PURE_ONCE_REWRITE_TAC
                                     [LEFT_IMP_FORALL_CONV
                                        (mk_imp (snd(dest_thm thm),w))]
                             THEN EXISTS_TAC tm2)
                    else ALL_TAC
            else ALL_TAC) in
      ASSUM_LIST (MAP_EVERY (\thm. f thm)) (asl,w)
      ? failwith 'SPEC_UNDISCH_MATCH_RHS_TAC';;
```

```
%-------------------------------------------------------------------

   File:          RANGE_INDUCT.ml

   Author:        (c) P. J. Windley 1993

   Description:

   Defines an induction tactic for use with ranges.

   Modified:

   23FEB93 (PJW) -- Original file.

-------------------------------------------------------------------%

%
let SUC_LESS_EQ = PROVE
   ("!m n. (SUC m) <= n ==> m <= n",
    REWRITE_TAC [LESS_OR_EQ]
    THEN REPEAT STRIP_TAC
    THENL [
IMP_RES_TAC SUC_LESS
THEN ASM_REWRITE_TAC []
    ;
    POP_ASSUM (SUBST1_TAC o GSYM)
    THEN REWRITE_TAC [LESS_SUC_REFL]
    ]
   );;
%

let SUC_LESS_EQ = mk_thm
   ([], "!m n. (SUC m) <= n ==> m <= n");;

let RANGE_PLUS_INDUCT_LEMMA = PROVE
   ("! P a b .
     (! u . (P(u + a) /\ SUC(u + a) <= b ==> P(SUC(u + a)))) /\
     ((a <= b) ==> P a) ==>
     (! u .(u + a) <= b ==> P (u + a))",
    REPEAT GEN_TAC
    THEN STRIP_TAC
    THEN IMP_RES_TAC SUC_LESS_EQ
    THEN INDUCT_TAC
    THEN ASM_REWRITE_TAC [ADD_CLAUSES]
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC SUC_LESS_EQ
    THEN RES_TAC
   );;

let RANGE_INDUCT_LEMMA = PROVE
   ("! P a b .
     (! u. (P(a + u) /\ SUC(a + u) <= b ==> P(SUC(a + u)))) /\
     ((a <= b) ==> P a) ==>
     !t' . a <= t' ==> t' <= b ==> P t'",
    REPEAT GEN_TAC
    THEN DISCH_THEN (STRIP_ASSUME_TAC o ONCE_REWRITE_RULE [ADD_SYM])
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC RANGE_PLUS_INDUCT_LEMMA
    THEN ASSUM_LIST (\asl . STRIP_ASSUME_TAC (
    REWRITE_RULE [MATCH_MP SUB_ADD (el 3 asl)]  (
    SPEC "t' - a" (el 1 asl))))
    THEN RES_TAC
   );;

let RANGE_INDUCT_TAC =
    MATCH_MP_TAC RANGE_INDUCT_LEMMA
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC SUC_LESS_EQ;;

%-------------------------------------------------------------------
```

```
Testing...

g "!t' . a <= t' ==> t' <= b ==> P t'";;
g "!t' . (a+1) <= t' ==> t' <= b ==> P t'";;
g "!t' . a <= t' ==> t' <= (b+1) ==> P t'";;
g "!t' . (a+1) <= t' ==> t' <= (b+1) ==> P t'";;
g "!t' . (a-1) <= t' ==> t' <= b ==> P t'";;


e(RANGE_INDUCT_TAC);;


-----------------------------------------------------------------%


%---------------------------------------------------------------------------------

  File:       assoc.ml

  Author:     (c) D.A. Fura 1992

  Date:       22 October 1992

  Associativity theorems involving + and -:

  ASSOC_ADD_ADD1: |- "! a b c . (a + b) + c = a + (b + c)"
  ASSOC_ADD_ADD2: |- "! a b c . (a + b) + c = a + (c + b)"
  ASSOC_ADD_ADD3: |- "! a b c . (a + b) + c = b + (a + c)"
  ASSOC_ADD_ADD4: |- "! a b c . (a + b) + c = b + (c + a)"
  ASSOC_ADD_ADD5: |- "! a b c . (a + b) + c = c + (a + b)"
  ASSOC_ADD_ADD6: |- "! a b c . (a + b) + c = c + (b + a)"
  ASSOC_ADD_SUB1: |- "! a b c . c <= b ==> ((a + b) - c = a + (b - c))"
  ASSOC_ADD_SUB2: |- "! a b c . c <= a ==> ((a + b) - c = b + (a - c))"
  ASSOC_ADD_SUB3: |- "! a b c . b <= c ==> ((a + b) - c = a - (c - b))"
  ASSOC_ADD_SUB4: |- "! a b c . a <= c ==> ((a + b) - c = b - (c - a))"
  ASSOC_SUB_ADD1: |- "! a b c . b <= a /\ b <= c ==> ((a - b) + c = a + (c - b))"
  ASSOC_SUB_ADD2: |- "! a b c . ((a - b) + c = c + (a - b))"
  ASSOC_SUB_ADD3: |- "! a b c . b <= a /\ c <= b ==> ((a - b) + c = a - (b - c))"
  ASSOC_SUB_ADD4: |- "! a b c . (a = b) ==> ((a - b) + c = c - (b - a))"
  ASSOC_SUB_SUB1: |- "! a b c . (a - b) - c = a - (b + c)"
  ASSOC_SUB_SUB2: |- "! a b c . (a - b) - c = a - (c + b)"

---------------------------------------------------------------------------------%

set_flag ('timing', true);;

system 'rm assoc.th';;

new_theory 'assoc';;

let SYM_RULE =
   (CONV_RULE (ONCE_DEPTH_CONV SYM_CONV))
? failwith 'SYM_RULE';;

let ASSOC_ADD_ADD1 = prove_thm
    ('ASSOC_ADD_ADD1',
     "! a b c . (a + b) + c = a + (b + c)",
     REWRITE_TAC [ADD_ASSOC]
    );;

let ASSOC_ADD_ADD2 = prove_thm
    ('ASSOC_ADD_ADD2',
     "! a b c . (a + b) + c = a + (c + b)",
     REPEAT GEN_TAC
     THEN SUBST_TAC [SPECL ["c:num";"b:num"] ADD_SYM]
     THEN REWRITE_TAC [ASSOC_ADD_ADD1]
    );;

let ASSOC_ADD_ADD3 = prove_thm
    ('ASSOC_ADD_ADD3',
     "! a b c . (a + b) + c = b + (a + c)",
      REPEAT GEN_TAC
      THEN SUBST_TAC [SPECL ["a:num";"b:num"] ADD_SYM]
      THEN REWRITE_TAC [ASSOC_ADD_ADD1]
     );;

let ASSOC_ADD_ADD4 = prove_thm
    ('ASSOC_ADD_ADD4',
     "! a b c . (a + b) + c = b + (c + a)",
     REPEAT GEN_TAC
     THEN SUBST_TAC [SPECL ["a:num";"b:num"] ADD_SYM]
     THEN REWRITE_TAC [ASSOC_ADD_ADD2]
    );;

let ASSOC_ADD_ADD5 = prove_thm
    ('ASSOC_ADD_ADD5',
     "! a b c . (a + b) + c = c + (a + b)",
     REWRITE_TAC [SPECL ["a+b";"c:num"] ADD_SYM]
    );;

let ASSOC_ADD_ADD6 = prove_thm
    ('ASSOC_ADD_ADD6',
     "! a b c . (a + b) + c = c + (b + a)",
     REPEAT GEN_TAC
     THEN SUBST_TAC [SPECL ["a:num";"b:num"] ADD_SYM]
     THEN REWRITE_TAC [SPECL ["b+a";"c:num"] ADD_SYM]
    );;

let ASSOC_ADD_SUB1 = prove_thm
    ('ASSOC_ADD_SUB1',
     "! a b c . c <= b ==> ((a + b) - c = a + (b - c))",
     REPEAT STRIP_TAC
     THEN IMP_RES_TAC LESS_EQ_ADD_SUB
     THEN ASM_REWRITE_TAC[]
    );;

let ASSOC_ADD_SUB2 = prove_thm
    ('ASSOC_ADD_SUB2',
     "! a b c . c <= a ==> ((a + b) - c = b + (a - c))",
      REPEAT GEN_TAC
      THEN SUBST_TAC [SPECL ["a:num";"b:num"] ADD_SYM]
      THEN REWRITE_TAC [ASSOC_ADD_SUB1]
    );;

let ASSOC_ADD_SUB3 = prove_thm
    ('ASSOC_ADD_SUB3',
     "! a b c . b <= c ==> ((a + b) - c = a - (c - b))",
      PURE_ONCE_REWRITE_TAC [LESS_OR_EQ]
      THEN REPEAT STRIP_TAC
      THENL [
         IMP_RES_TAC LESS_ADD_1
         THEN ASM_REWRITE_TAC[]
         THEN SUBST_TAC [SPECL ["a+b";"b:num";"p+1"] SUB_PLUS]
         THEN ASSUME_TAC (SPEC "b:num" LESS_EQ_REFL)
         THEN IMP_RES_TAC (SPECL ["a:num";"b:num";"b:num"] ASSOC_ADD_SUB1)
         THEN SUBST_TAC [SPECL ["b:num";"p+1"] ADD_SYM]
         THEN IMP_RES_TAC (SPECL ["p+1";"b:num";"b:num"] ASSOC_ADD_SUB1)
         THEN ASM_REWRITE_TAC [SUB_EQUAL_0; ADD_CLAUSES]
      ;
         ASSUME_TAC (SPEC "c:num" LESS_EQ_REFL)
         THEN IMP_RES_TAC (SPECL ["a:num";"c:num";"c:num"] ASSOC_ADD_SUB1)
         THEN ASM_REWRITE_TAC [SUB_EQUAL_0; ADD_CLAUSES; SUB_0]
      ]
    );;

let ASSOC_ADD_SUB4 = prove_thm
    ('ASSOC_ADD_SUB4',
     "! a b c . a <= c ==> ((a + b) - c = b - (c - a))",
     REPEAT GEN_TAC
     THEN SUBST_TAC [SPECL ["a:num";"b:num"] ADD_SYM]
     THEN REWRITE_TAC [ASSOC_ADD_SUB3]
    );;
```

```
let ASSOC_SUB_ADD1 = prove_thm
   ('ASSOC_SUB_ADD1',
    "! a b c . b <= a /\ b <= c ==> ((a - b) + c = a + (c - b))",
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC (SYM_RULE (SPECL ["c:num";"a:num";"b:num"] ASSOC_ADD_SUB2))
    THEN ASM_REWRITE_TAC[]
    THEN IMP_RES_TAC (SPECL ["c:num";"a:num";"b:num"] ASSOC_ADD_SUB1)
    THEN ONCE_ASM_REWRITE_TAC[]
    THEN SUBST_TAC [SPECL ["c:num";"a-b"] ADD_SYM]
    THEN REWRITE_TAC[]
   );;

let ASSOC_SUB_ADD2 = prove_thm
   ('ASSOC_SUB_ADD2',
    "! a b c . ((a - b) + c = c + (a - b))",
    REPEAT GEN_TAC
    THEN SUBST_TAC [SPECL ["a-b";"c:num"] ADD_SYM]
    THEN REWRITE_TAC[]
   );;

let ASSOC_SUB_ADD3 = prove_thm
   ('ASSOC_SUB_ADD3',
    "! a b c . b <= a /\ c <= b ==> ((a - b) + c = a - (b - c))",
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC (SYM_RULE (SPECL ["c:num";"a:num";"b:num"] ASSOC_ADD_SUB4))
    THEN ASM_REWRITE_TAC[]
    THEN IMP_RES_TAC (SPECL ["c:num";"a:num";"b:num"] ASSOC_ADD_SUB1)
    THEN ASM_REWRITE_TAC [SPECL ["c:num";"a-b"] ADD_SYM]
   );;

let ASSOC_SUB_ADD4 = prove_thm
   ('ASSOC_SUB_ADD4',
    "! a b c . (a = b) ==> ((a - b) + c = c - (b - a))",
    REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC [SUB_EQUAL_0; ADD_CLAUSES; SUB_0]
   );;

let ASSOC_SUB_SUB1 = prove_thm
   ('ASSOC_SUB_SUB1',
    "! a b c . (a - b) - c = a - (b + c)",
    REWRITE_TAC [SUB_PLUS]
   );;

let ASSOC_SUB_SUB2 = prove_thm
   ('ASSOC_SUB_SUB2',
    "! a b c . (a - b) - c = a - (c + b)",
    REPEAT GEN_TAC
    THEN SUBST_TAC [SPECL ["c:num";"b:num"] ADD_SYM]
    THEN REWRITE_TAC [SUB_PLUS]
   );;

close_theory();;


%-------------------------------------------------------------------------------

  File:       cond.ml

  Author:     (c) D.A. Fura 1992

  Date:       30 December 1992

  Theorems involving conditionals:

  COND_TRUE_TRUE: |- "! a b c d.  a => (a => b | c) | d = (a => b | d)"
  COND_TRUE_FALSE: |- "! a b c d.  a => ((~a) => b | c) | d = (a => c | d)"
  COND_FALSE_TRUE: |- "! a b c d.  a => b | (a => c | d) = (a => b | c)"
  COND_FALSE_FALSE: |- "! a b c d.  a => b | (a => c | d) = (a => b | d)"
  COND_TRUE_CHOICES: |- "!a. (a => T | T) = T"
  COND_FALSE_CHOICES: |- "!a. (a => F | F) = F"
  COND_FIRST_CHOICE: |- "!a b c. (b => a | c = a) /\ ~(a = c) ==> b"
  COND_SECOND_CHOICE: |- "!a b c. (b => a | c = c) /\ ~(a = c) ==> ~b"
```

```
--------------------------------------------------------------------------------%

set_flag ('timing', true);;

system 'rm cond.th';;

new_theory 'cond';;

%[PJW]%
let SYM_RULE =
  (CONV_RULE (ONCE_DEPTH_CONV SYM_CONV))
? failwith 'SYM_RULE';;

let COND_TRUE_TRUE = prove_thm
   ('COND_TRUE_TRUE',
    "! (a :bool) (b c d :*) .  a => (a => b | c) | d = (a => b | d)",
    REPEAT GEN_TAC
    THEN BOOL_CASES_TAC "a:bool"
    THEN REWRITE_TAC[]
   );;

let COND_TRUE_FALSE = prove_thm
   ('COND_TRUE_FALSE',
    "! (a :bool) (b c d :*) .  a => ((~a) => b | c) | d = (a => c | d)",
    REPEAT GEN_TAC
    THEN BOOL_CASES_TAC "a:bool"
    THEN REWRITE_TAC[]
   );;

let COND_FALSE_TRUE = prove_thm
   ('COND_FALSE_TRUE',
    "! (a :bool) (b c d :*) .  a => b | ((~a) => c | d) = (a => b | c)",
    REPEAT GEN_TAC
    THEN BOOL_CASES_TAC "a:bool"
    THEN REWRITE_TAC[]
   );;

let COND_FALSE_FALSE = prove_thm
   ('COND_FALSE_FALSE',
    "! (a :bool) (b c d :*) .  a => b | (a => c | d) = (a => b | d)",
    REPEAT GEN_TAC
    THEN BOOL_CASES_TAC "a:bool"
    THEN REWRITE_TAC[]
   );;

let COND_TRUE_CHOICES = prove_thm
   ('COND_TRUE_CHOICES',
    "! (a :bool) . a => T | T = T",
    REPEAT GEN_TAC
    THEN BOOL_CASES_TAC "a:bool"
    THEN REWRITE_TAC[]
   );;

let COND_FALSE_CHOICES = prove_thm
   ('COND_FALSE_CHOICES',
    "! (a :bool) . a => F | F = F",
    REPEAT GEN_TAC
    THEN BOOL_CASES_TAC "a:bool"
    THEN REWRITE_TAC[]
   );;

let COND_FIRST_CHOICE = prove_thm
   ('COND_FIRST_CHOICE',
    "! (a c :*) (b :bool) .  (b => a | c = a) /\ ~(a = c) ==> b",
    REPEAT STRIP_TAC
    THEN UNDISCH_TAC "(b:bool) => (a:*) | c = a"
    THEN ASM_CASES_TAC "b:bool"
    THEN POP_ASSUM_LIST
       (MAP_EVERY (\thm. ASSUME_TAC thm THEN ASSUME_TAC (SYM_RULE thm)))
    THEN ASM_REWRITE_TAC[]
   );;
```

```
let COND_SECOND_CHOICE = prove_thm
   ('COND_SECOND_CHOICE',
    "! (a c :*) (b :bool) .  (b => a | c = c) /\ ~(a = c) ==> (~b)",
    REPEAT STRIP_TAC
    THEN UNDISCH_TAC "(b:bool) => (a:*) | c = c"
    THEN ASM_CASES_TAC "b:bool"
    THEN ASM_REWRITE_TAC[]
    THEN RES_TAC
   );;

close_theory();;


%------------------------------------------------------------------------------

  File:        ineq.ml

  Author:      (c) D.A. Fura 1992-93

  Date:        19 February 1993

  Theorems involving inequalities:

  NOT_EQ: |- "! m n. ~(m = n) = ((m < n) \/ (n < m))"
  NOT_EQ_ZERO: |- "! n . ~(n = 0) = (n > 0)"
  NOT_LESS_EQ_LESS [PL]: |- "!m n. (~(m <= n)) = (n < m)"

  LT_IMP_SUC_LE: |- "! m n. m < n ==> SUC m <= n"
  SUC_LE_IMP_LT: |- "! m n. SUC m <= n ==> m < n"
  LT_EQ_SUC_LE: |- "! m n. m < n = SUC m <= n"
  LT_IMP_LE_PRE: |- "! m n. m < n ==> m <= PRE n"
  LE_PRE_IMP_LT: |- "! m n. 1 <= n ==> (m <= PRE n ==> m < n)"
  LT_EQ_LE_PRE: |- "! m n. 1 <= n ==> (m <= PRE n ==> m < n)"
  LT_EQ_LE_PRE: |- "! m n. 1 <= n ==> (m < n = m <= PRE n)"
  LE_IMP_LT_SUC: |- "! m n. m <= n ==> m < SUC n"
  LT_SUC_IMP_LE: |- "! m n. m < SUC n ==> m <= n"
  LE_EQ_LT_SUC: |- "! m n. m <= n = m < SUC n"
  LE_IMP_PRE_LT: |- "! m n. 1 <= m ==> (m <= n ==> PRE m < n)"
  PRE_LT_IMP_LE: |- "! m n. PRE m < n ==> m <= n"

  LT_IMP_LE: |- "! m n. m < n ==> (m <= n)"
  SUC_LE_IMP_LE: |- "! m n. (m + 1) <= n ==> (m <= n)"

  LESS_EQ_ZERO [PJW]: |- "! n. (n <= 0) = (n = 0)"
  ONE_LESS_EQ: |- "! n. (1 <= n) = (n > 0)"
  LESS_THAN_ONE: |- "! n . n < 1 = (n = 0)"

  LESS_EQ_MONO_SUB: |- "!m n p. (m <= n) ==> ((m - p) <= (n - p))"
  LESS_EQ_MONO_SUB_EQ:
     |- "! m n p .  (p <= m) /\ (p <= n) ==> ((m - p) <= (n - p) = m <= n)"

  LESS_EQ_ADD_SUB1: |- "! n p . (n <= p) ==> !m. (((m + n) <= p) = (m <= p - n))"
  LESS_EQ_ADD_SUB2: |- "! m p . (m <= p) ==> !n. (((m + n) <= p) = (n <= p - m))"
  LESS_EQ_SUB_ADD1: |- "! m n . (n <= m) ==> !p. ((m - n) <= p = m <= (n + p))"
  LESS_EQ_SUB_ADD2: |- "! m n . (n <= m) ==> !p. ((m - n) <= p = m <= (p + n))"

  LESS_LESS_EQ_TRANS [WP]: |- "! m n p . (m < n) /\ (n <= p) ==> (m < p)"
  LESS_EQ_LESS_TRANS [WP]: |- "! m n p . (m <= n) /\ (n < p) ==> (m < p)"

  LESS_EQ_3_CASES: |- "n <= 3 = (((n = 0) \/ (n = 1)) \/ (n = 2)) \/ (n = 3)"
  LESS_EQ_15_CASES:
     |- "n <= 15 =
         ((((((((((((((n = 0) \/ (n = 1)) \/ (n = 2)) \/ (n = 3)) \/ (n = 4)) \/
         (n = 5)) \/ (n = 6)) \/ (n = 7)) \/ (n = 8)) \/ (n = 9)) \/ (n = 10)) \/
         (n = 11)) \/ (n = 12)) \/ (n = 13)) \/ (n = 14)) \/ (n = 15)

  [PL] = (c) Paul Loewenstein
  [WP] = (c) Wim Ploegarts
  [PJW] = (c) Phil Windley.

------------------------------------------------------------------------------%
```

```
set_flag ('timing', true);;

system 'rm ineq.th';;

new_theory 'ineq';;

load_library 'reduce';;

let SYM_RULE =
  (CONV_RULE (ONCE_DEPTH_CONV SYM_CONV))
? failwith 'SYM_RULE';;

let LIMP = (\thm. GEN_ALL (fst (EQ_IMP_RULE (SPEC_ALL thm))));;

let NOT_EQ = prove_thm
   ('NOT_EQ',
    "! (m n :num) . ~(m = n) = ((m < n) \/ (n < m))",
    INDUCT_TAC
    THEN INDUCT_TAC
    THEN REDUCE_TAC
    THEN ASM_REWRITE_TAC [LESS_0;SYM_RULE NOT_SUC;SYM_RULE SUC_NOT;LESS_MONO_EQ;
                          ADD1;EQ_MONO_ADD_EQ]
   );;

let NOT_EQ_ZERO = prove_thm
   ('NOT_EQ_ZERO',
    "! (n :num) . ~(n = 0) = (n > 0)",
    INDUCT_TAC
    THEN REDUCE_TAC
    THEN REWRITE_TAC [NOT_SUC;GREATER;LESS_0]
   );;

% [PJW] %
let LESS_EQ_ZERO = prove_thm
   ('LESS_EQ_ZERO',
    "! (n :num) . (n <= 0) = (n = 0)",
    REWRITE_TAC [GREATER_OR_EQ; LESS_OR_EQ; NOT_LESS_0]
   );;

let ONE_LESS_EQ = prove_thm
   ('ONE_LESS_EQ',
    "! (n :num) . (1 <= n) = (n > 0)",
    REWRITE_TAC [GREATER;LESS_EQ;SYM (num_CONV "1")]
   );;

%<PL>%
let NOT_LESS_EQ_LESS = prove_thm
   ('NOT_LESS_EQ_LESS',
    "!m n. (~(m <= n)) = (n < m)",
     REWRITE_TAC [SYM (SPEC_ALL NOT_LESS)]
   );;

let LT_IMP_SUC_LE = prove_thm
   ('LT_IMP_SUC_LE',
    "! m n :num. m < n ==> SUC m <= n", ACCEPT_TAC LESS_OR );;

let SUC_LE_IMP_LT = prove_thm
   ('SUC_LE_IMP_LT',
    "! m n :num. SUC m <= n ==> m < n", ACCEPT_TAC OR_LESS );;

let LT_EQ_SUC_LE = prove_thm
   ('LT_EQ_SUC_LE',
    "! m n :num. m < n = SUC m <= n", ACCEPT_TAC LESS_EQ );;

let LT_IMP_LE_PRE = prove_thm
   ('LT_IMP_LE_PRE',
    "! m n :num. m < n ==> m <= PRE n",
    REWRITE_TAC [PRE_SUB1]
    THEN REPEAT GEN_TAC
    THEN ACCEPT_TAC (SPECL ["n:num";"m:num"] SUB_LESS_OR)
   );;
```

```
let LE_PRE_IMP_LT = prove_thm
   ('LE_PRE_IMP_LT',
    "! m n :num. 1 <= n ==> (m <= PRE n ==> m < n)",
    REWRITE_TAC
       [PRE_SUB1;SYM_RULE (SPECL ["m:num";"n-1";"1"] LESS_EQ_MONO_ADD_EQ)]
    THEN REPEAT GEN_TAC
    THEN DISCH_TAC
    THEN IMP_RES_TAC (SPECL ["n:num";"1"] SUB_ADD)
    THEN ASM_REWRITE_TAC[REWRITE_RULE [ADD1] SUC_LE_IMP_LT]
   );;

let LT_EQ_LE_PRE = prove_thm
   ('LT_EQ_LE_PRE',
    "! m n :num. 1 <= n ==> (m < n = m <= PRE n)",
    REPEAT GEN_TAC
    THEN DISCH_TAC
    THEN EQ_TAC
    THENL [
       ACCEPT_TAC (SPEC_ALL LT_IMP_LE_PRE)
    ;
       IMP_RES_TAC LE_PRE_IMP_LT
       THEN ASM_REWRITE_TAC[]
    ]
   );;

let LE_IMP_LT_SUC = prove_thm
   ('LE_IMP_LT_SUC',
    "! m n :num. m <= n ==> m < SUC n",
    ACCEPT_TAC LESS_EQ_IMP_LESS_SUC
   );;

let LT_SUC_IMP_LE = prove_thm
   ('LT_SUC_IMP_LE',
    "! m n :num. m < SUC n ==> m <= n",
     REWRITE_TAC [ADD1;LESS_OR_EQ]
     THEN REPEAT STRIP_TAC
     THEN ASM_CASES_TAC "m:num = n"
     THEN ASM_REWRITE_TAC[]
     THEN IMP_RES_TAC (LIMP NOT_EQ)
     THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
     THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LT_IMP_SUC_LE)
     THEN IMP_RES_TAC LESS_EQ_ANTISYM
   );;

let LE_EQ_LT_SUC = prove_thm
   ('LE_EQ_LT_SUC',
    "! m n :num. m <= n = m < SUC n",
    REPEAT GEN_TAC
    THEN EQ_TAC
    THENL [
       ACCEPT_TAC (SPECL ["m:num";"n:num"] LE_IMP_LT_SUC)
    ;
       ACCEPT_TAC (SPEC_ALL LT_SUC_IMP_LE)
    ]
   );;

let LESS_THAN_ONE = prove_thm
   ('LESS_THAN_ONE',
    "! (n :num) . n < 1 = (n = 0)",
    REWRITE_TAC [SYM (SPEC "n:num" LESS_EQ_ZERO)]
    THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "1<=1"))
    THEN IMP_RES_TAC
          (REWRITE_RULE [PRE_SUB1] (SPECL ["n:num";"1"] LT_EQ_LE_PRE))
    THEN ASM_REWRITE_TAC [REDUCE_CONV "1-1"]
   );;

let LE_IMP_PRE_LT = prove_thm
   ('LE_IMP_PRE_LT',
    "! m n :num. 1 <= m ==> (m <= n ==> PRE m < n)",
    REWRITE_TAC [PRE_SUB1]
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LE_IMP_LT_SUC)
    THEN REWRITE_TAC [SYM_RULE (SPECL ["m-1";"n:num";"1"] LESS_MONO_ADD_EQ)]
    THEN IMP_RES_TAC SUB_ADD
    THEN ASM_REWRITE_TAC[]
   );;

let PRE_LT_IMP_LE = prove_thm
   ('PRE_LT_IMP_LE',
    "! m n :num. PRE m < n ==> m <= n",
    REPEAT GEN_TAC
    THEN ASM_CASES_TAC "1 <= m"
    THEN REWRITE_TAC
           [PRE_SUB1;SYM_RULE (SPECL ["m-1";"n:num";"1"] LESS_MONO_ADD_EQ)]
    THEN IMP_RES_TAC SUB_ADD
    THEN ASM_REWRITE_TAC [REWRITE_RULE [ADD1] LE_EQ_LT_SUC]
    THEN IMP_RES_TAC NOT_LESS_EQ_LESS
    THEN IMP_RES_TAC LESS_THAN_ONE
    THEN ASSUME_TAC
          (SYM_RULE
            (REWRITE_RULE [ADD1] (SPECL ["0";"n:num"] LE_EQ_LT_SUC)))
    THEN ASM_REWRITE_TAC [ZERO_LESS_EQ]
   );;

let LT_IMP_LE = prove_thm
   ('LT_IMP_LE',
    "! (m n :num) . m < n ==> (m <= n)",
    REWRITE_TAC [LESS_OR_EQ]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let SUC_LE_IMP_LE = prove_thm
   ('SUC_LE_IMP_LE',
    "! (m n :num) . (m + 1) <= n ==> (m <= n)",
    REPEAT STRIP_TAC
    THEN ASSUME_TAC (SPECL ["m:num";"1"] LESS_EQ_ADD)
    THEN IMP_RES_TAC LESS_EQ_TRANS
   );;

let lemma1 = TAC_PROOF
   (([], "! m n p :num .
          (m <= n) ==> (~m < p) ==> (~n < p)"),
    REWRITE_TAC [NOT_LESS]
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC (SPECL ["p:num";"m:num";"n:num"] LESS_EQ_TRANS)
   );;

let LESS_EQ_MONO_SUB = prove_thm
   ('LESS_EQ_MONO_SUB',
    "!m n p. (m <= n) ==> ((m - p) <= (n - p))",
    INDUCT_TAC
    THEN INDUCT_TAC
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC [SUB_0; ZERO_LESS_EQ]
    THENL [
       UNDISCH_TAC "(SUC m) <= 0"
       THEN ASM_REWRITE_TAC [LESS_EQ_ZERO]
       THEN DISCH_TAC
       THEN ASM_REWRITE_TAC[]
       THEN ASM_REWRITE_TAC[SUB]
    ;
       ASM_REWRITE_TAC [SUB]
       THEN COND_CASES_TAC
       THENL [
          REWRITE_TAC [ZERO_LESS_EQ]
       ;
          ASSUME_TAC (SPECL ["m:num";"n:num"] LESS_EQ_MONO)
          THEN ASSUME_TAC (SPECL ["m-p";"n-p"] LESS_EQ_MONO)
          THEN RES_TAC
          THEN RES_TAC
          THEN IMP_RES_TAC (SPECL ["m:num";"n:num";"p:num"] lemma1)
          THEN ASM_REWRITE_TAC[]
```

```
        ]
      ]
    );;

let LESS_EQ_MONO_SUB_EQ = prove_thm
    ('LESS_EQ_MONO_SUB_EQ',
     "! m n p :num .
       (p <= m) /\ (p <= n) ==> ((m - p) <= (n - p) = m <= n)",
     REPEAT STRIP_TAC
     THEN IMP_RES_TAC (SPECL ["n:num";"p:num"] SUB_ADD)
     THEN IMP_RES_TAC (SPECL ["m:num";"p:num"] SUB_ADD)
     THEN ASM_REWRITE_TAC
              [SPECL ["m-p";"n-p";"p:num"] (SYM_RULE LESS_EQ_MONO_ADD_EQ)]
    );;

let LESS_EQ_ADD_SUB1 = prove_thm
    ('LESS_EQ_ADD_SUB1',
     "! n p :num . (n <= p) ==> !m:num. (((m + n) <= p) = (m <= p - n))",
     REPEAT STRIP_TAC
     THEN PURE_REWRITE_TAC [SPECL ["m:num";"p:num - n:num";"n:num"]
                              (SYM_RULE LESS_EQ_MONO_ADD_EQ)]
     THEN IMP_RES_TAC (SPECL ["n:num";"p:num"] SUB_ADD)
     THEN ASM_REWRITE_TAC[]
    );;

let LESS_EQ_ADD_SUB2 = prove_thm
    ('LESS_EQ_ADD_SUB2',
     "! m p :num . (m <= p) ==> !n:num. (((m + n) <= p) = (n <= p - m))",
     REPEAT STRIP_TAC
     THEN PURE_REWRITE_TAC [SPECL ["n:num";"p:num - m:num";"m:num"]
                              (SYM_RULE LESS_EQ_MONO_ADD_EQ)]
     THEN SUBST_TAC [SPECL ["n:num";"m:num"] ADD_SYM]
     THEN IMP_RES_TAC (SPECL ["p:num";"m:num"] SUB_ADD)
     THEN ASM_REWRITE_TAC[]
    );;

let LESS_EQ_SUB_ADD1 = prove_thm
    ('LESS_EQ_SUB_ADD1',
     "! m n :num . (n <= m) ==> !p:num. ((m - n) <= p = m <= (n + p))",
     REPEAT STRIP_TAC
     THEN SUBST_TAC [SYM (SPECL ["m-n";"p:num";"n:num"] LESS_EQ_MONO_ADD_EQ)]
     THEN SUBST_TAC [(SPECL ["p:num";"n:num"] ADD_SYM)]
     THEN IMP_RES_TAC (SPECL ["m:num";"n:num"] SUB_ADD)
     THEN ASM_REWRITE_TAC[]
    );;

let LESS_EQ_SUB_ADD2 = prove_thm
    ('LESS_EQ_SUB_ADD2',
     "! m n :num . (n <= m) ==> !p:num. ((m - n) <= p = m <= (p + n))",
     REPEAT STRIP_TAC
     THEN SUBST_TAC [SYM (SPECL ["m-n";"p:num";"n:num"] LESS_EQ_MONO_ADD_EQ)]
     THEN IMP_RES_TAC (SPECL ["m:num";"n:num"] SUB_ADD)
     THEN ASM_REWRITE_TAC[]
    );;

%WP 4-9-90%
let LESS_LESS_EQ_TRANS = prove_thm
    ('LESS_LESS_EQ_TRANS',
     "! m n p . (m < n) /\ (n <= p) ==> (m < p)",
     REWRITE_TAC [LESS_OR_EQ] THEN
     REPEAT STRIP_TAC THEN
     IMP_RES_TAC LESS_TRANS THEN
     ASM_REWRITE_TAC [] THEN
     FIRST_ASSUM \thm . (SUBST_TAC [SYM thm]  ? NO_TAC) THEN
     FIRST_ASSUM ACCEPT_TAC
    );;

%WP 4-9-90%
let LESS_EQ_LESS_TRANS = prove_thm
    ('LESS_EQ_LESS_TRANS',
     "! m n p . (m <= n) /\ (n < p) ==> (m < p)",
     REWRITE_TAC [LESS_OR_EQ] THEN
    REPEAT STRIP_TAC THEN
    IMP_RES_TAC LESS_TRANS THEN
    ASM_REWRITE_TAC [] THEN
    FIRST_ASSUM \thm . (SUBST_TAC [SYM thm]  ? NO_TAC) THEN
    FIRST_ASSUM ACCEPT_TAC
   );;

letrec LESS_CASES_NCONV i_tm =
   let tm = mk_const((string_of_int i_tm),":num") in
   if i_tm = 1 then SPEC "n:num" LESS_THAN_ONE
   else
     REWRITE_RULE
      [LESS_CASES_NCONV (i_tm - 1)]
      (REWRITE_RULE
        [SPECL ["n:num";mk_const((string_of_int (i_tm - 1)),":num")] LESS_OR_EQ]
        (REWRITE_RULE
          [REWRITE_RULE [] (REDUCE_CONV (list_mk_comb ("$-",[tm;"1"])))]
          (SUBS_OCCS
            [([2],
              (MP
                (REWRITE_RULE [ONE_LESS_EQ;PRE_SUB1]
                              (SPECL ["n:num";tm] LT_EQ_LE_PRE))
                (REWRITE_RULE [] (REDUCE_CONV (list_mk_comb ("$>",[tm;"0"]))))))]
            (REFL (list_mk_comb ("$<",["n:num";tm]))))));;

let LESS_EQ_CASES_CONV tm =
   let i_tm = int_of_string (fst (dest_const tm)) in
   if i_tm = 0 then SPEC "n:num" LESS_EQ_ZERO
   else
     REWRITE_RULE
       [LESS_CASES_NCONV i_tm]
       (SPECL ["n:num";tm] LESS_OR_EQ);;

let LESS_EQ_1_CASES = save_thm
   ('LESS_EQ_1_CASES',
     LESS_EQ_CASES_CONV "1"
   );;

let LESS_EQ_3_CASES = save_thm
   ('LESS_EQ_3_CASES',
     LESS_EQ_CASES_CONV "3"
   );;

let LESS_EQ_15_CASES = save_thm
   ('LESS_EQ_15_CASES',
     LESS_EQ_CASES_CONV "15"
   );;

close_theory();;
```

3 Design Verification

This section contains the HOL listings for the PIU design verification—clock level with respect to gate level. The five subsections of this section describe the P-Port, M-Port, R-Port, C-Port, and SU-Cont, respectively.

3.1 P-Port Clock-Level Verification

The theory *pclock_ver* and file *pc_thms.ml* contain the P-Port clock-level correctness proof.

```
%-------------------------------------------------------------------------------

   File:          pclock_ver.ml

   Author:        (c) D.A. Fura 1992-93

   Date:          1 March 1993

-------------------------------------------------------------------------------%

set_flag ('timing', true);;

set_search_path (search_path() @ ['/home/elvis6/dfura/ftep/piu/hol/pport/';
                                  '/home/elvis6/dfura/ftep/piu/hol/lib/';
                                  '/home/elvis6/dfura/hol/ml/';
                                  '/home/elvis6/dfura/hol/Library/abs_theory/';
                                  '/home/elvis6/dfura/hol/Library/time/';
                                  '/home/elvis6/dfura/hol/Library/GI/';
                                  '/home/elvis6/dfura/hol/Library/tools/'
                                 ]);;

system 'rm pclock_ver.th';;

new_theory 'pclock_ver';;

loadf 'aux_defs';;

load_library 'reduce';;

map new_parent
     ['gates_def1';'latches_def';'ffs_def';'counters_def';'ineq'];;

map load_parent ['pclock_def';'pblock_def';'paux_def';'piuaux_def';'busn_def';
                 'buses_def';'array_def';'wordn_def';'assoc';'cond'];;

let PC_OF_EXP = theorem 'pclock_def' 'PC_OF_EXP';;
let PC_NSF_EXP = theorem 'pclock_def' 'PC_NSF_EXP';;

loadt 'pc_thms.ml';;

let P_Clock_Correct = prove_thm
   ('P_Clock_Correct',
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       PCSet_Correct s e p",
    REPEAT STRIP_TAC
    THEN REWRITE_TAC [PCSet_Correct]
    THEN INDUCT_THEN (prove_induction_thm PCI) ASSUME_TAC
    THEN GEN_TAC
    THEN REWRITE_TAC [PC_Correct;PC_Exec;PC_PreC;PC_PostC]
    THEN CONJ_TAC
    THENL [
       % Subgoal 1: "s(t + 1) = PC_NSF(s t)(e t)" %
```

```
        SUBST_TAC [SPEC "(s (t+1)):pc_state" State_Selectors_Work]
        THEN IMP_RES_TAC (SYM_RULE P_addrS_THM)
        THEN IMP_RES_TAC (SYM_RULE P_dest1S_THM)
        THEN IMP_RES_TAC (SYM_RULE P_be_S_THM)
        THEN IMP_RES_TAC (SYM_RULE P_wrS_THM)
        THEN IMP_RES_TAC (SYM_RULE P_fsm_stateS_THM)
        THEN IMP_RES_TAC (SYM_RULE P_fsm_rstS_THM)
        THEN IMP_RES_TAC (SYM_RULE P_fsm_mrqtS_THM)
        THEN IMP_RES_TAC (SYM_RULE P_fsm_sackS_THM)
        THEN IMP_RES_TAC (SYM_RULE P_fsm_cgnt_S_THM)
        THEN IMP_RES_TAC (SYM_RULE P_fsm_crqt_S_THM)
        THEN IMP_RES_TAC (SYM_RULE P_fsm_hold_S_THM)
        THEN IMP_RES_TAC (SYM_RULE P_fsm_lock_S_THM)
        THEN IMP_RES_TAC (SYM_RULE P_rqtS_THM)
        THEN IMP_RES_TAC (SYM_RULE P_sizeS_THM)
        THEN IMP_RES_TAC (SYM_RULE P_loadS_THM)
        THEN IMP_RES_TAC (SYM_RULE P_downS_THM)
        THEN IMP_RES_TAC (SYM_RULE P_lock_S_THM)
        THEN IMP_RES_TAC (SYM_RULE P_lock_inh_S_THM)
        THEN IMP_RES_TAC (SYM_RULE P_male_S_THM)
        THEN IMP_RES_TAC (SYM_RULE P_rale_S_THM)
        THEN ASM_REWRITE_TAC
              [SPEC "PC_NSF ((s:time->pc_state) t) ((e:time->pc_env) t)"
                    (SYM_RULE State_Selectors_Work)]
      ;
        % Subgoal 2: "p t = PC_OF(s t)(e t)" %
        SUBST_TAC [SPEC "((p:time->pc_out) t)" Out_Selectors_Work]
        THEN IMP_RES_TAC (SYM_RULE L_ad_outO_THM)
        THEN IMP_RES_TAC (SYM_RULE L_ready_O_THM)
        THEN IMP_RES_TAC (SYM_RULE I_ad_outO_THM)
        THEN IMP_RES_TAC (SYM_RULE I_be_O_THM)
        THEN IMP_RES_TAC (SYM_RULE I_rale_O_THM)
        THEN IMP_RES_TAC (SYM_RULE I_male_O_THM)
        THEN IMP_RES_TAC (SYM_RULE I_crqt_O_THM)
        THEN IMP_RES_TAC (SYM_RULE I_cale_O_THM)
        THEN IMP_RES_TAC (SYM_RULE I_mrdy_O_THM)
        THEN IMP_RES_TAC (SYM_RULE I_last_O_THM)
        THEN IMP_RES_TAC (SYM_RULE I_hlda_O_THM)
        THEN IMP_RES_TAC (SYM_RULE I_lock_O_THM)
        THEN ASM_REWRITE_TAC
              [SPEC "PC_OF ((s:time->pc_state) t) ((e:time->pc_env) t)"
                    (SYM_RULE Out_Selectors_Work)]
      ]
   );;

close_theory();;


%------------------------------------------------------------------------------

   File:          pc_thms.ml

   Author:        (c) D.A. Fura 1992-93

   Date:          1 March 1993

------------------------------------------------------------------------------%

let P_addrS_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
        PBlock_GATE s e p
           ==>
        (P_addrS (PC_NSF (s t) (e t)) = P_addrS (s (t+1)))"),
    REWRITE_TAC [P_addrS;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let P_dest1S_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
```

```
      PBlock_GATE s e p
         ==>
      (P_dest1S (PC_NSF (s t) (e t)) = P_dest1S (s (t+1)))"),
   REWRITE_TAC [P_dest1S;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC[]
  );;

let P_be_S_THM = TAC_PROOF
  (([],
   "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
      PBlock_GATE s e p
         ==>
      (P_be_S (PC_NSF (s t) (e t)) = P_be_S (s (t+1)))"),
   REWRITE_TAC [P_be_S;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC[]
  );;

let P_wrS_THM = TAC_PROOF
  (([],
   "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
      PBlock_GATE s e p
         ==>
      (P_wrS (PC_NSF (s t) (e t)) = P_wrS (s (t+1)))"),
   REWRITE_TAC [P_wrS;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC[]
  );;

let P_fsm_stateS_THM = TAC_PROOF
  (([],
   "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
      PBlock_GATE s e p
         ==>
      (P_fsm_stateS (PC_NSF (s t) (e t)) = P_fsm_stateS (s (t+1)))"),
   REWRITE_TAC [P_fsm_stateS;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC[]
  );;

let P_fsm_rstS_THM = TAC_PROOF
  (([],
   "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
      PBlock_GATE s e p
         ==>
      (P_fsm_rstS (PC_NSF (s t) (e t)) = P_fsm_rstS (s (t+1)))"),
   REWRITE_TAC [P_fsm_rstS;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC[]
  );;

let P_fsm_mrqtS_THM = TAC_PROOF
  (([],
   "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
      PBlock_GATE s e p
         ==>
      (P_fsm_mrqtS (PC_NSF (s t) (e t)) = P_fsm_mrqtS (s (t+1)))"),
   REWRITE_TAC [P_fsm_mrqtS;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC[]
  );;

let P_fsm_sackS_THM = TAC_PROOF
  (([],
   "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
      PBlock_GATE s e p
         ==>
      (P_fsm_sackS (PC_NSF (s t) (e t)) = P_fsm_sackS (s (t+1)))"),
   REWRITE_TAC [P_fsm_sackS;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC[]
```

```
      );;

let P_fsm_cgnt_S_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (P_fsm_cgnt_S (PC_NSF (s t) (e t)) = P_fsm_cgnt_S (s (t+1)))"),
    REWRITE_TAC [P_fsm_cgnt_S;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let P_fsm_crqt_S_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (P_fsm_crqt_S (PC_NSF (s t) (e t)) = P_fsm_crqt_S (s (t+1)))"),
    REWRITE_TAC [P_fsm_crqt_S;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let P_fsm_hold_S_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (P_fsm_hold_S (PC_NSF (s t) (e t)) = P_fsm_hold_S (s (t+1)))"),
    REWRITE_TAC [P_fsm_hold_S;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let P_fsm_lock_S_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (P_fsm_lock_S (PC_NSF (s t) (e t)) = P_fsm_lock_S (s (t+1)))"),
    REWRITE_TAC [P_fsm_lock_S;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let P_rqtS_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (P_rqtS (PC_NSF (s t) (e t)) = P_rqtS (s (t+1)))"),
    REWRITE_TAC [P_rqtS;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let P_sizeS_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (P_sizeS (PC_NSF (s t) (e t)) = P_sizeS (s (t+1)))"),
    REWRITE_TAC [P_sizeS;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let P_loadS_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
```

```
         ==>
      (P_loadS (PC_NSF (s t) (e t)) = P_loadS (s (t+1)))"),
    REWRITE_TAC [P_loadS;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let P_downS_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (P_downS (PC_NSF (s t) (e t)) = P_downS (s (t+1)))"),
    REWRITE_TAC [P_downS;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let P_lock_S_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (P_lock_S (PC_NSF (s t) (e t)) = P_lock_S (s (t+1)))"),
    REWRITE_TAC [P_lock_S;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let lemma1 = TAC_PROOF
   (([],
     "SUBARRAY
       (MALTER
        (MALTER
         (ALTER
          (ALTER
           (MALTER
             ARBN
             (31,28)
             ((~P_rqtS(s (t:time))) => FST(L_be_E(e t)) | P_be_S(s t)))
           27
           ((~P_rqtS(s t)) => FST(L_wrE(e t)) | P_wrS(s t)))
          26
          F)
         (25,24)
         (SUBARRAY
          ((~P_rqtS(s t)) =>
           SUBARRAY(FST(L_ad_inE(e t)))(25,0) |
           P_addrS(s t))
          (1,0)))
        (23,0)
        (SUBARRAY
         ((~P_rqtS(s t)) =>
          SUBARRAY(FST(L_ad_inE(e t)))(25,0) |
          P_addrS(s t))
         (25,2)))
       (23,22)
      =
      SUBARRAY
       ((~P_rqtS(s t)) =>
        SUBARRAY(FST(L_ad_inE(e t)))(25,0) |
        P_addrS(s t))
       (25,24)"),
     CONV_TAC (ONCE_DEPTH_CONV FUN_EQ_CONV)
     THEN ASM_REWRITE_TAC [ALTER_THM;MALTER_THM;SUBARRAY_THM]
     THEN GEN_TAC
     THEN ASSUME_TAC (SPEC "n+22" ZERO_LESS_EQ)
     THEN ASSUME_TAC (SPEC "2" ZERO_LESS_EQ)
     THEN IMP_RES_TAC (SPECL ["n+22";"0";"2"] ASSOC_SUB_ADD1)
     THEN ASM_REWRITE_TAC [ZERO_LESS_EQ; COND_TRUE_TRUE;
                           SYM (SPECL ["n+22";"23";"2"] LESS_EQ_MONO_ADD_EQ);
                           SPECL ["n:num";"22";"2"] ASSOC_ADD_ADD1]
      THEN REDUCE_TAC
      THEN ASM_REWRITE_TAC [COND_TRUE_TRUE;
                            SPECL ["n:num";"22";"2"] ASSOC_ADD_ADD1]
      THEN REDUCE_TAC
      THEN ASM_REWRITE_TAC [COND_TRUE_TRUE]
     );;

let P_lock_inh_S_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (P_lock_inh_S (PC_NSF (s t) (e t)) = P_lock_inh_S (s (t+1)))"),
    REWRITE_TAC [P_lock_inh_S;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1]
   );;

let P_male_S_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (P_male_S (PC_NSF (s t) (e t)) = P_male_S (s (t+1)))"),
    REWRITE_TAC [P_male_S;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC [lemma1]
   );;

let P_rale_S_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (P_rale_S (PC_NSF (s t) (e t)) = P_rale_S (s (t+1)))"),
    REWRITE_TAC [P_rale_S;PBlock_EXP;(EXPAND_LET_RULE PC_NSF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC [lemma1]
   );;

let L_ad_outO_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (L_ad_outO (PC_OF (s t) (e t)) = L_ad_outO (p t))"),
    REWRITE_TAC [L_ad_outO;PBlock_EXP;(EXPAND_LET_RULE PC_OF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC []
   );;

let L_ready_O_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (L_ready_O (PC_OF (s t) (e t)) = L_ready_O (p t))"),
    REWRITE_TAC [L_ready_O;PBlock_EXP;(EXPAND_LET_RULE PC_OF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC []
   );;

let lemma2 = TAC_PROOF
   (([],
     "(fsm_astate t = ((P_fsm_stateS (s (t+1)) = PA),
                       (P_fsm_stateS (s (t+1)) = PA))) ==>
       (fsm_dstate t = ((P_fsm_stateS (s (t+1)) = PD),
                        (P_fsm_stateS (s (t+1)) = PD))) ==>
        (data_out_en t = ((FST(wr_out t) /\ FST(fsm_dstate t)),
                          (SND(wr_out t) /\ SND(fsm_dstate t)))) ==>
         (ad_addr_out t =
            ((FST(fsm_astate t) => BUSN(FST(addr_out t)) | Offn),
            (SND(fsm_astate t) => BUSN(SND(addr_out t)) | Offn))) ==>
         (ad_data_out t =
            ((FST(data_out_en t) => BUSN(FST(data_out t)) | Offn),
             (SND(data_out_en t) => BUSN(SND(data_out t)) | Offn))) ==>
          (Bus2n_CF (31,0) (ad_data_out t) (ad_addr_out t) = T)"),
    REWRITE_TAC [EXPAND_LET_RULE Bus2n_CF;ASel;BSel]
    THEN REPEAT STRIP_TAC
    THEN ASM_CASES_TAC "P_fsm_stateS(s(t + 1)) = PD"
    THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "0<=31"))
    THEN IMP_RES_TAC OFFnP_Offn
    THEN IMP_RES_TAC OFFnP_BUSN
    THEN ASM_REWRITE_TAC[]
    THENL [
       ASM_CASES_TAC "FST((wr_out:time->bool#bool) t)"
    ;
       ASM_CASES_TAC "SND((wr_out:time->bool#bool) t)"
    ]
    THEN ASM_REWRITE_TAC [SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
   );;

let NOT_PD = TAC_PROOF
   (([], "! (x :pfsm_ty) . ~(x = PD) ==> ((x = PA) \/ (x = PH))"),
    INDUCT_THEN (prove_induction_thm pfsm_ty_Axiom) ASSUME_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let I_ad_outO_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (I_ad_outO (PC_OF (s t) (e t)) = I_ad_outO (p t))"),
    REWRITE_TAC [I_ad_outO;PBlock_EXP;(EXPAND_LET_RULE PC_OF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN POP_ASSUM_LIST (MAP_EVERY (STRIP_ASSUME_TAC o SPEC_ALL))
    THEN IMP_RES_TAC lemma2
    THEN ASM_REWRITE_TAC [ONnP_BUSN;ONnP_Offn;COND_TRUE_TRUE]
    THEN ASM_CASES_TAC
       "((P_fsm_rstS(s (t:time)) => PA |
         ((P_fsm_stateS(s t) = PH) =>
          (P_fsm_hold_S(s t) => PA | PH) |
         ((P_fsm_stateS(s t) = PA) =>
         ((P_fsm_mrqtS(s t) \/ ~P_fsm_crqt_S(s t) /\ ~P_fsm_cgnt_S(s t)) => PD |
          ((~P_fsm_hold_S(s t) /\ P_fsm_lock_S(s t)) => PH | PA)) |
         ((P_fsm_sackS(s t) /\ P_fsm_hold_S(s t)) => PA |
          ((P_fsm_sackS(s t) /\ ~P_fsm_hold_S(s t) /\ ~P_fsm_lock_S(s t)) => PA |
           ((P_fsm_sackS(s t) /\ ~P_fsm_hold_S(s t) /\ P_fsm_lock_S(s t)) => PH |
            PD)))))) = PD)"
    THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "0<=31"))
    THEN IMP_RES_TAC ONnP_Offn
    THEN ASM_REWRITE_TAC [ONnP_BUSN;COND_TRUE_TRUE]
    THEN IMP_RES_TAC NOT_PD
    THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
    THEN ASM_REWRITE_TAC [ONnP_BUSN;ONnP_Offn;COND_TRUE_TRUE;
                          prove_constructors_distinct pfsm_ty_Axiom;
                          SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
    THEN COND_CASES_TAC
    THEN ASM_REWRITE_TAC [ONnP_BUSN;ONnP_Offn;COND_TRUE_TRUE;
                          prove_constructors_distinct pfsm_ty_Axiom;
                          SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
   );;

let I_be_O_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (I_be_O (PC_OF (s t) (e t)) = I_be_O (p t))"),
    REWRITE_TAC [I_be_O;PBlock_EXP;(EXPAND_LET_RULE PC_OF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC []
   );;
```

```
let I_rale_O_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (I_rale_O (PC_OF (s t) (e t)) = I_rale_O (p t))"),
    REWRITE_TAC [I_rale_O;PBlock_EXP;(EXPAND_LET_RULE PC_OF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC [lemma1]
   );;

let I_male_O_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (I_male_O (PC_OF (s t) (e t)) = I_male_O (p t))"),
    REWRITE_TAC [I_male_O;PBlock_EXP;(EXPAND_LET_RULE PC_OF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC [lemma1]
   );;

let I_crqt_O_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (I_crqt_O (PC_OF (s t) (e t)) = I_crqt_O (p t))"),
    REWRITE_TAC [I_crqt_O;PBlock_EXP;(EXPAND_LET_RULE PC_OF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC []
   );;

let I_cale_O_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (I_cale_O (PC_OF (s t) (e t)) = I_cale_O (p t))"),
    REWRITE_TAC [I_cale_O;PBlock_EXP;(EXPAND_LET_RULE PC_OF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC []
   );;

let I_mrdy_O_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (I_mrdy_O (PC_OF (s t) (e t)) = I_mrdy_O (p t))"),
    REWRITE_TAC [I_mrdy_O;PBlock_EXP;(EXPAND_LET_RULE PC_OF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC [WIRE]
   );;

let I_last_O_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
       (I_last_O (PC_OF (s t) (e t)) = I_last_O (p t))"),
    REWRITE_TAC [I_last_O;PBlock_EXP;(EXPAND_LET_RULE PC_OF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC []
   );;

let I_hlda_O_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
       PBlock_GATE s e p
          ==>
```

```
      (I_hlda_O (PC_OF (s t) (e t)) = I_hlda_O (p t))"),
    REWRITE_TAC [I_hlda_O;PBlock_EXP;(EXPAND_LET_RULE PC_OF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC []
   );;

let I_lock_O_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->pc_state) (e :time->pc_env) (p :time->pc_out) .
      PBlock_GATE s e p
         ==>
      (I_lock_O (PC_OF (s t) (e t)) = I_lock_O (p t))"),
    REWRITE_TAC [I_lock_O;PBlock_EXP;(EXPAND_LET_RULE PC_OF_EXP)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC [lemma1]
   );;
```

3.2 M-Port Clock-Level Verification

The theory *mclock_ver* and file *mc_thms.ml* contain the M-Port clock-level correctness proof.

```
%-------------------------------------------------------------------------------

   File:          mclock_ver.ml

   Author:        (c) D.A. Fura 1992-93

   Date:          1 March 1993

-------------------------------------------------------------------------------%

set_flag ('timing', true);;

set_search_path (search_path() @ ['/home/elvis6/dfura/ftep/piu/hol/mport/';
                                  '/home/elvis6/dfura/ftep/piu/hol/lib/';
                                  '/home/elvis6/dfura/hol/ml/';
                                  '/home/elvis6/dfura/hol/Library/abs_theory/';
                                  '/home/elvis6/dfura/hol/Library/tools/'
                                 ]);;

system 'rm mclock_ver.th';;

new_theory 'mclock_ver';;

loadf 'abs_theory';;
loadf 'aux_defs';;

map new_parent ['wordn_def';'array_def';'gates_def1';'latches_def';'ffs_def';
                'counters_def';'ineq'];;
map load_parent ['piuaux_def';'maux_def';'mblock_def';'mclock_def'];;

let REP_ty = abs_type_info (theorem 'piuaux_def' 'REP');;

new_type_abbrev ('timeC', ":num");;

loadt 'mc_thms.ml';;

let M_Clock_Correct = prove_thm
   ('M_Clock_Correct',
    "! (rep :^REP_ty) (t :time) (s :time->mc_state) (e :time->mc_env)
       (p :time->mc_out) .
       MBlock_GATE rep s e p
          ==>
       MCSet_Correct rep s e p",
    REPEAT STRIP_TAC
    THEN REWRITE_TAC [MCSet_Correct]
```

```
     THEN INDUCT_THEN (prove_induction_thm MCI) ASSUME_TAC
     THEN GEN_TAC
     THEN REWRITE_TAC [MC_Correct;MC_Exec;MC_PreC;MC_PostC]
     THEN CONJ_TAC
     THENL [
        % Subgoal 1: "s(t + 1) = MC_NSF rep (s t)(e t)" %
        SUBST_TAC [SPEC "(s (t+1)):mc_state" State_Selectors_Work]
        THEN IMP_RES_TAC (SYM_RULE M_fsm_stateS_THM)
        THEN IMP_RES_TAC (SYM_RULE M_fsm_male_S_THM)
        THEN IMP_RES_TAC (SYM_RULE M_fsm_rdS_THM)
        THEN IMP_RES_TAC (SYM_RULE M_fsm_bwS_THM)
        THEN IMP_RES_TAC (SYM_RULE M_fsm_wwS_THM)
        THEN IMP_RES_TAC (SYM_RULE M_fsm_last_S_THM)
        THEN IMP_RES_TAC (SYM_RULE M_fsm_mrdy_S_THM)
        THEN IMP_RES_TAC (SYM_RULE M_fsm_zero_cntS_THM)
        THEN IMP_RES_TAC (SYM_RULE M_fsm_rstS_THM)
        THEN IMP_RES_TAC (SYM_RULE M_seS_THM)
        THEN IMP_RES_TAC (SYM_RULE M_wrS_THM)
        THEN IMP_RES_TAC (SYM_RULE M_addrS_THM)
        THEN IMP_RES_TAC (SYM_RULE M_beS_THM)
        THEN IMP_RES_TAC (SYM_RULE M_countS_THM)
        THEN IMP_RES_TAC (SYM_RULE M_rdyS_THM)
        THEN IMP_RES_TAC (SYM_RULE M_wwdelS_THM)
        THEN IMP_RES_TAC (SYM_RULE M_parityS_THM)
        THEN IMP_RES_TAC (SYM_RULE M_rd_dataS_THM)
        THEN IMP_RES_TAC (SYM_RULE M_detectS_THM)
        THEN ASM_REWRITE_TAC
              [SPEC "MC_NSF (rep:^REP_ty) ((s:time->mc_state) t)
                             ((e:time->mc_env) t)"
                    (SYM_RULE State_Selectors_Work)]
     ;
        % Subgoal 2: "p t = MC_OF rep (s t)(e t)" %
        SUBST_TAC [SPEC "((p:time->mc_out) t)" Out_Selectors_Work]
        THEN IMP_RES_TAC (SYM_RULE I_ad_outO_THM)
        THEN IMP_RES_TAC (SYM_RULE I_srdy_O_THM)
        THEN IMP_RES_TAC (SYM_RULE MB_addrO_THM)
        THEN IMP_RES_TAC (SYM_RULE MB_data_outO_THM)
        THEN IMP_RES_TAC (SYM_RULE MB_cs_eeprom_O_THM)
        THEN IMP_RES_TAC (SYM_RULE MB_cs_sram_O_THM)
        THEN IMP_RES_TAC (SYM_RULE MB_we_O_THM)
        THEN IMP_RES_TAC (SYM_RULE MB_oe_O_THM)
        THEN IMP_RES_TAC (SYM_RULE MB_parityO_THM)
        THEN ASM_REWRITE_TAC
              [SPEC "MC_OF (rep:^REP_ty) ((s:time->mc_state) t)
                            ((e:time->mc_env) t)"
                    (SYM_RULE Out_Selectors_Work)]
     ]
   );;

close_theory();;


%----------------------------------------------------------------------------

   File:          mc_thms.ml

   Author:        (c) D.A. Fura 1992-93

   Date:          1 March 1993

----------------------------------------------------------------------------%

let M_fsm_stateS_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
       MBlock_GATE rep s e p
          ==>
       (M_fsm_stateS (MC_NSF rep (s t) (e t)) = M_fsm_stateS (s (t+1)))"),
    REWRITE_TAC [M_fsm_stateS;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let M_fsm_male_S_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
       MBlock_GATE rep s e p
          ==>
       (M_fsm_male_S (MC_NSF rep (s t) (e t)) = M_fsm_male_S (s (t+1)))"),
    REWRITE_TAC [M_fsm_male_S;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let M_fsm_rdS_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
       MBlock_GATE rep s e p
          ==>
       (M_fsm_rdS (MC_NSF rep (s t) (e t)) = M_fsm_rdS (s (t+1)))"),
    REWRITE_TAC [M_fsm_rdS;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let M_fsm_bwS_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
       MBlock_GATE rep s e p
          ==>
       (M_fsm_bwS (MC_NSF rep (s t) (e t)) = M_fsm_bwS (s (t+1)))"),
    REWRITE_TAC [M_fsm_bwS;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let M_fsm_wwS_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
       MBlock_GATE rep s e p
          ==>
       (M_fsm_wwS (MC_NSF rep (s t) (e t)) = M_fsm_wwS (s (t+1)))"),
    REWRITE_TAC [M_fsm_wwS;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let M_fsm_last_S_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
       MBlock_GATE rep s e p
          ==>
       (M_fsm_last_S (MC_NSF rep (s t) (e t)) = M_fsm_last_S (s (t+1)))"),
    REWRITE_TAC [M_fsm_last_S;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let M_fsm_mrdy_S_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
       MBlock_GATE rep s e p
          ==>
       (M_fsm_mrdy_S (MC_NSF rep (s t) (e t)) = M_fsm_mrdy_S (s (t+1)))"),
    REWRITE_TAC [M_fsm_mrdy_S;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
    );;

let M_fsm_zero_cntS_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
      MBlock_GATE rep s e p
         ==>
       (M_fsm_zero_cntS (MC_NSF rep (s t) (e t)) = M_fsm_zero_cntS (s (t+1)))"),
    REWRITE_TAC [M_fsm_zero_cntS;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let M_fsm_rstS_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
      MBlock_GATE rep s e p
         ==>
       (M_fsm_rstS (MC_NSF rep (s t) (e t)) = M_fsm_rstS (s (t+1)))"),
    REWRITE_TAC [M_fsm_rstS;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let M_seS_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
      MBlock_GATE rep s e p
         ==>
       (M_seS (MC_NSF rep (s t) (e t)) = M_seS (s (t+1)))"),
    REWRITE_TAC [M_seS;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let M_wrS_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
      MBlock_GATE rep s e p
         ==>
       (M_wrS (MC_NSF rep (s t) (e t)) = M_wrS (s (t+1)))"),
    REWRITE_TAC [M_wrS;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let M_addrS_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
      MBlock_GATE rep s e p
         ==>
       (M_addrS (MC_NSF rep (s t) (e t)) = M_addrS (s (t+1)))"),
    REWRITE_TAC [M_addrS;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let M_beS_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
      MBlock_GATE rep s e p
         ==>
       (M_beS (MC_NSF rep (s t) (e t)) = M_beS (s (t+1)))"),
    REWRITE_TAC [M_beS;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let M_countS_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
       MBlock_GATE rep s e p
          ==>
       (M_countS (MC_NSF rep (s t) (e t)) = M_countS (s (t+1)))"),
    REWRITE_TAC [M_countS;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let M_rdyS_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
       MBlock_GATE rep s e p
          ==>
       (M_rdyS (MC_NSF rep (s t) (e t)) = M_rdyS (s (t+1)))"),
    REWRITE_TAC [M_rdyS;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let M_wwdelS_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
       MBlock_GATE rep s e p
          ==>
       (M_wwdelS (MC_NSF rep (s t) (e t)) = M_wwdelS (s (t+1)))"),
    REWRITE_TAC [M_wwdelS;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let M_parityS_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
       MBlock_GATE rep s e p
          ==>
       (M_parityS (MC_NSF rep (s t) (e t)) = M_parityS (s (t+1)))"),
    REWRITE_TAC [M_parityS;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let M_rd_dataS_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
       MBlock_GATE rep s e p
          ==>
       (M_rd_dataS (MC_NSF rep (s t) (e t)) = M_rd_dataS (s (t+1)))"),
    REWRITE_TAC [M_rd_dataS;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let M_detectS_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
       MBlock_GATE rep s e p
          ==>
       (M_detectS (MC_NSF rep (s t) (e t)) = M_detectS (s (t+1)))"),
    REWRITE_TAC [M_detectS;MBlock_EXP;(EXPAND_LET_RULE MC_NSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let I_ad_outO_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
      MBlock_GATE rep s e p
         ==>
       (I_ad_outO (MC_OF rep (s t) (e t)) = I_ad_outO (p t))"),
    REWRITE_TAC [I_ad_outO;MBlock_EXP;(EXPAND_LET_RULE MC_OF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let I_srdy_O_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
      MBlock_GATE rep s e p
         ==>
       (I_srdy_O (MC_OF rep (s t) (e t)) = I_srdy_O (p t))"),
    REWRITE_TAC [I_srdy_O;MBlock_EXP;(EXPAND_LET_RULE MC_OF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let MB_addrO_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
      MBlock_GATE rep s e p
         ==>
       (MB_addrO (MC_OF rep (s t) (e t)) = MB_addrO (p t))"),
    REWRITE_TAC [MB_addrO;MBlock_EXP;(EXPAND_LET_RULE MC_OF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let MB_data_outO_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
      MBlock_GATE rep s e p
         ==>
       (MB_data_outO (MC_OF rep (s t) (e t)) = MB_data_outO (p t))"),
    REWRITE_TAC [MB_data_outO;MBlock_EXP;(EXPAND_LET_RULE MC_OF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let MB_cs_eeprom_O_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
      MBlock_GATE rep s e p
         ==>
       (MB_cs_eeprom_O (MC_OF rep (s t) (e t)) = MB_cs_eeprom_O (p t))"),
    REWRITE_TAC [MB_cs_eeprom_O;MBlock_EXP;(EXPAND_LET_RULE MC_OF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let MB_cs_sram_O_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
      MBlock_GATE rep s e p
         ==>
       (MB_cs_sram_O (MC_OF rep (s t) (e t)) = MB_cs_sram_O (p t))"),
    REWRITE_TAC [MB_cs_sram_O;MBlock_EXP;(EXPAND_LET_RULE MC_OF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let MB_we_O_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
       MBlock_GATE rep s e p
          ==>
       (MB_we_O (MC_OF rep (s t) (e t)) = MB_we_O (p t))"),
    REWRITE_TAC [MB_we_O;MBlock_EXP;(EXPAND_LET_RULE MC_OF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let MB_oe_O_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
       MBlock_GATE rep s e p
          ==>
       (MB_oe_O (MC_OF rep (s t) (e t)) = MB_oe_O (p t))"),
    REWRITE_TAC [MB_oe_O;MBlock_EXP;(EXPAND_LET_RULE MC_OF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let MB_parityO_THM = TAC_PROOF
   (([],
    "! (t :timeC) (rep :^REP_ty) (s :timeC->mc_state) (e :timeC->mc_env)
       (p :timeC->mc_out) .
       MBlock_GATE rep s e p
          ==>
       (MB_parityO (MC_OF rep (s t) (e t)) = MB_parityO (p t))"),
    REWRITE_TAC [MB_parityO;MBlock_EXP;(EXPAND_LET_RULE MC_OF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;
```

3.3 R-Port Clock-Level Verification

The theory *rclock_ver* and file *rc_thms.ml* contain the R-Port clock-level correctness proof.

```
%----------------------------------------------------------------------------

   File:          rclock_ver.ml

   Author:        (c) D.A. Fura 1992-93

   Date:          7 March 1993

----------------------------------------------------------------------------%

set_flag ('timing', true);;

set_search_path (search_path() @ ['/home/elvis6/dfura/ftep/piu/hol/rport/';
                                  '/home/elvis6/dfura/ftep/piu/hol/lib/';
                                  '/home/elvis6/dfura/hol/ml/';
                                  '/home/elvis6/dfura/hol/Library/abs_theory/';
                                  '/home/elvis6/dfura/hol/Library/tools/'
                                 ]);;

system 'rm rclock_ver.th';;

new_theory 'rclock_ver';;

loadf 'abs_theory';;
loadf 'aux_defs';;

map new_parent ['array_def';'rclock_def';'rblock_def'];;
map load_parent ['piuaux_def';'raux_def';'buses_def';'busn_def';'ineq';'cond';
                 'wordn_def'];;

load_library 'reduce';;

let REP_ty = abs_type_info (theorem 'piuaux_def' 'REP');;

let RClockNSF_REW = theorem 'rclock_def' 'RClockNSF_REW';;
let RClockOF_REW = theorem 'rclock_def' 'RClockOF_REW';;
let RBlock_EXP = theorem 'rblock_def' 'RBlock_EXP';;

let RClockNSF_EXP = EXPAND_LET_RULE RClockNSF_REW;;
let RClockOF_EXP = EXPAND_LET_RULE RClockOF_REW;;
%
loadt 'rc_thms.ml';;

let R_Clock_Correct = prove_thm
   ('R_Clock_Correct',
    "! (rep :^REP_ty) (t :time) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       RCSet_Correct rep s e p",
    REPEAT STRIP_TAC
    THEN REWRITE_TAC [RCSet_Correct]
    THEN INDUCT_THEN (prove_induction_thm RCI) ASSUME_TAC
    THEN GEN_TAC
    THEN REWRITE_TAC [RC_Correct;RC_Exec;RC_PreC;RC_PostC]
    THEN CONJ_TAC
    THENL [
       % Subgoal 1: "s(t + 1) = RClockNSF rep (s t)(e t)" %
       SUBST_TAC [SPEC "(s (t+1)):r_state" State_Selectors_Work]
       THEN IMP_RES_TAC (SYM_RULE R_ctr0_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr1_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr2_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr3_THM)
       THEN IMP_RES_TAC (SYM_RULE R_busA_latch_THM)
       THEN IMP_RES_TAC (SYM_RULE R_fsm_state_THM)
       THEN IMP_RES_TAC (SYM_RULE R_fsm_ale_THM)
       THEN IMP_RES_TAC (SYM_RULE R_fsm_mrdy_THM)
       THEN IMP_RES_TAC (SYM_RULE R_fsm_last_THM)
       THEN IMP_RES_TAC (SYM_RULE R_fsm_rst_THM)
       THEN IMP_RES_TAC (SYM_RULE R_int0_dis_THM)
       THEN IMP_RES_TAC (SYM_RULE R_int3_dis_THM)
       THEN IMP_RES_TAC (SYM_RULE R_c01_cout_del_THM)
       THEN IMP_RES_TAC (SYM_RULE R_int1_en_THM)
       THEN IMP_RES_TAC (SYM_RULE R_c23_cout_del_THM)
       THEN IMP_RES_TAC (SYM_RULE R_int2_en_THM)
       THEN IMP_RES_TAC (SYM_RULE R_wr_THM)
       THEN IMP_RES_TAC (SYM_RULE R_cntlatch_del_THM)
       THEN IMP_RES_TAC (SYM_RULE R_srdy_del_THM)
       THEN IMP_RES_TAC (SYM_RULE R_reg_sel_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr0_in_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr0_mux_sel_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr0_irden_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr0_cry_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr0_new_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr0_out_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr0_orden_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr1_in_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr1_mux_sel_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr1_irden_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr1_cry_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr1_new_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr1_out_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr1_orden_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr2_in_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr2_mux_sel_THM)
       THEN IMP_RES_TAC (SYM_RULE R_ctr2_irden_THM)
```

```
        THEN IMP_RES_TAC (SYM_RULE R_ctr2_cry_THM)
        THEN IMP_RES_TAC (SYM_RULE R_ctr2_new_THM)
        THEN IMP_RES_TAC (SYM_RULE R_ctr2_out_THM)
        THEN IMP_RES_TAC (SYM_RULE R_ctr2_orden_THM)
        THEN IMP_RES_TAC (SYM_RULE R_ctr3_in_THM)
        THEN IMP_RES_TAC (SYM_RULE R_ctr3_mux_sel_THM)
        THEN IMP_RES_TAC (SYM_RULE R_ctr3_irden_THM)
        THEN IMP_RES_TAC (SYM_RULE R_ctr3_cry_THM)
        THEN IMP_RES_TAC (SYM_RULE R_ctr3_new_THM)
        THEN IMP_RES_TAC (SYM_RULE R_ctr3_out_THM)
        THEN IMP_RES_TAC (SYM_RULE R_ctr3_orden_THM)
        THEN IMP_RES_TAC (SYM_RULE R_icr_load_THM)
        THEN IMP_RES_TAC (SYM_RULE R_icr_old_THM)
        THEN IMP_RES_TAC (SYM_RULE R_icr_mask_THM)
        THEN IMP_RES_TAC (SYM_RULE R_icr_THM)
        THEN IMP_RES_TAC (SYM_RULE R_icr_rden_THM)
        THEN IMP_RES_TAC (SYM_RULE R_ccr_THM)
        THEN IMP_RES_TAC (SYM_RULE R_ccr_rden_THM)
        THEN IMP_RES_TAC (SYM_RULE R_gcr_THM)
        THEN IMP_RES_TAC (SYM_RULE R_gcr_rden_THM)
        THEN IMP_RES_TAC (SYM_RULE R_sr_THM)
        THEN IMP_RES_TAC (SYM_RULE R_sr_rden_THM)
        THEN ASM_REWRITE_TAC
              [SPEC "RClockNSF (rep:^REP_ty) ((s:time->r_state) t)
                            ((e:time->r_env) t)"
                   (SYM_RULE State_Selectors_Work)]
     ;
        % Subgoal 2: "p t = RClockOF rep (s t)(e t)" %
        SUBST_TAC [SPEC "((p:time->r_out) t)" Out_Selectors_Work]
        THEN IMP_RES_TAC (SYM_RULE I_ad_out_THM)
        THEN IMP_RES_TAC (SYM_RULE I_srdy_THM)
        THEN IMP_RES_TAC (SYM_RULE Int0_THM)
        THEN IMP_RES_TAC (SYM_RULE Int1_THM)
        THEN IMP_RES_TAC (SYM_RULE Int2_THM)
        THEN IMP_RES_TAC (SYM_RULE Int3_THM)
        THEN IMP_RES_TAC (SYM_RULE Ccr_THM)
        THEN IMP_RES_TAC (SYM_RULE Led_THM)
        THEN IMP_RES_TAC (SYM_RULE Reset_error_THM)
        THEN IMP_RES_TAC (SYM_RULE Pmm_invalid_THM)
        THEN ASM_REWRITE_TAC
              [SPEC "RClockOF (rep:^REP_ty) ((s:time->r_state) t)
                           ((e:time->r_env) t)"
                   (SYM_RULE Out_Selectors_Work)]
     ]
   );;

close_theory();;


%----------------------------------------------------------------------------

   File:          rc_thms.ml

   Author:        (c) D.A. Fura 1992-93

   Date:          7 March 1993

----------------------------------------------------------------------------%

let lemma1 = TAC_PROOF
   (([], "! (a b :bool) . a /\ (b \/ a) = a"),
    REPEAT GEN_TAC
    THEN BOOL_CASES_TAC "a:bool"
    THEN REWRITE_TAC[]
   );;

let lemma1_spec = SPECL ["((R_fsm_rstS((s:time->r_state) t) => RI |
                          ((R_fsm_stateS(s t) = RI) =>
                          ((~R_fsm_ale_S(s t)) => RA | RI) |
                          ((R_fsm_stateS(s t) = RA) =>
                          ((~R_fsm_mrdy_S(s t)) => RD | RA) |
                          ((~R_fsm_last_S(s t)) => RI | RA)))) = RD)";
```

```
                                   "((R_fsm_rstS((s:time->r_state) t) => RI |
                                    ((R_fsm_stateS(s t) = RI) =>
                                    ((~R_fsm_ale_S(s t)) => RA | RI) |
                                    ((R_fsm_stateS(s t) = RA) =>
                                    ((~R_fsm_mrdy_S(s t)) => RD | RA) |
                                    ((~R_fsm_last_S(s t)) => RI | RA)))) = RA)"]
                                  lemma1;;

let lemma2 = TAC_PROOF
   (([], "! (a :rfsm_ty) . ~(a = RD) /\ ((a = RA) \/ (a = RD)) = (a = RA)"),
    GEN_TAC
    THEN ASM_CASES_TAC "a = RD"
    THEN ASM_REWRITE_TAC [SYM_RULE (prove_constructors_distinct rfsm_ty_Axiom)]
   );;

let lemma2_spec = SPEC "(R_fsm_rstS((s:time->r_state) t) => RI |
                         ((R_fsm_stateS(s t) = RI) =>
                         ((~R_fsm_ale_S(s t)) => RA | RI) |
                         ((R_fsm_stateS(s t) = RA) =>
                         ((~R_fsm_mrdy_S(s t)) => RD | RA) |
                         ((~R_fsm_last_S(s t)) => RI | RA))))"
                        lemma2;;

let lemma3 = TAC_PROOF
   (([], "! (a b :bool) . a \/ b \/ a = b \/ a"),
    REPEAT GEN_TAC
    THEN BOOL_CASES_TAC "a:bool"
    THEN BOOL_CASES_TAC "b:bool"
    THEN REWRITE_TAC[]
   );;

let EXPANSION_EQ tm =
   let leqtm =
          "((~R_srdy_del_S(s(t - 1))) =>
           INCN 3(R_reg_selS(s(t - 1))) |
           R_reg_selS(s(t - 1)))" in
   let rconjtm = mk_eq (leqtm, tm) in
   let lconjtm =
          "(~((~SND(I_rale_E(e(t - 1)))) =>
           ELEMENT(SND(I_ad_inE(e(t - 1))))27 |
           R_wrS(s(t - 1))) /\
           ((R_fsm_rstS(s(t - 1)) =>
             RI |
             ((R_fsm_stateS(s(t - 1)) = RI) =>
              ((~R_fsm_ale_S(s(t - 1))) => RA | RI) |
              ((R_fsm_stateS(s(t - 1)) = RA) =>
               ((~R_fsm_mrdy_S(s(t - 1))) => RD | RA) |
               ((~R_fsm_last_S(s(t - 1))) => RI | RA)))) =
           RA))" in
   mk_conj (lconjtm, rconjtm);;

let WORDN_NOT_EQ_THMLIST tm =
   let l0 = if tm = "0" then []
            else [REDUCE_RULE (SPECL ["0";tm] WORDN_3_NOT_EQUAL)] in
   let l1 = if tm = "1" then l0
            else l0 @ [REDUCE_RULE (SPECL ["1";tm] WORDN_3_NOT_EQUAL)] in
   let l2 = if tm = "2" then l1
            else l1 @ [REDUCE_RULE (SPECL ["2";tm] WORDN_3_NOT_EQUAL)] in
   let l3 = if tm = "3" then l2
            else l2 @ [REDUCE_RULE (SPECL ["3";tm] WORDN_3_NOT_EQUAL)] in
   let l4 = if tm = "4" then l3
            else l3 @ [REDUCE_RULE (SPECL ["4";tm] WORDN_3_NOT_EQUAL)] in
   let l5 = if tm = "5" then l4
            else l4 @ [REDUCE_RULE (SPECL ["5";tm] WORDN_3_NOT_EQUAL)] in
   let l6 = if tm = "6" then l5
            else l5 @ [REDUCE_RULE (SPECL ["6";tm] WORDN_3_NOT_EQUAL)] in
   let l7 = if tm = "7" then l6
            else l6 @ [REDUCE_RULE (SPECL ["7";tm] WORDN_3_NOT_EQUAL)] in
   let l8 = if tm = "8" then l7
            else l7 @ [REDUCE_RULE (SPECL ["8";tm] WORDN_3_NOT_EQUAL)] in
   let l9 = if tm = "9" then l8
            else l8 @ [REDUCE_RULE (SPECL ["9";tm] WORDN_3_NOT_EQUAL)] in
   let l10 = if tm = "10" then l9
             else l9 @ [REDUCE_RULE (SPECL ["10";tm] WORDN_3_NOT_EQUAL)] in
   let l11 = if tm = "11" then l10
             else l10 @ [REDUCE_RULE (SPECL ["11";tm] WORDN_3_NOT_EQUAL)] in
   let l12 = if tm = "12" then l11
             else l11 @ [REDUCE_RULE (SPECL ["12";tm] WORDN_3_NOT_EQUAL)] in
   let l13 = if tm = "13" then l12
             else l12 @ [REDUCE_RULE (SPECL ["13";tm] WORDN_3_NOT_EQUAL)] in
   let l14 = if tm = "14" then l13
             else l13 @ [REDUCE_RULE (SPECL ["14";tm] WORDN_3_NOT_EQUAL)] in
   let l15 = if tm = "15" then l14
             else l14 @ [REDUCE_RULE (SPECL ["15";tm] WORDN_3_NOT_EQUAL)] in
   l15;;

let lemmaCF = TAC_PROOF
   (([],"! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
           (p :time->r_out) .
         (RBlock_GATE rep s e p) /\ (t > 0)
           ==>
         (Bus12n_CF
           (31,0)
           ((R_ctr0_irdenS(s t) => BUSN(R_ctr0_inS(s t)) | Offn),Offn)
           ((R_ctr0_ordenS(s t) => BUSN(R_ctr0_outS(s t)) | Offn),Offn)
           ((R_ctr1_irdenS(s t) => BUSN(R_ctr1_inS(s t)) | Offn),Offn)
           ((R_ctr1_ordenS(s t) => BUSN(R_ctr1_outS(s t)) | Offn),Offn)
           ((R_ctr2_irdenS(s t) => BUSN(R_ctr2_inS(s t)) | Offn),Offn)
           ((R_ctr2_ordenS(s t) => BUSN(R_ctr2_outS(s t)) | Offn),Offn)
           ((R_ctr3_irdenS(s t) => BUSN(R_ctr3_inS(s t)) | Offn),Offn)
           ((R_ctr3_ordenS(s t) => BUSN(R_ctr3_outS(s t)) | Offn),Offn)
           ((R_icr_rdenS(s t) => BUSN(R_icrS(s t)) | Offn),Offn)
           ((R_ccr_rdenS(s t) => BUSN(R_ccrS(s t)) | Offn),Offn)
           ((R_gcr_rdenS(s t) => BUSN(R_gcrS(s t)) | Offn),Offn)
           ((R_sr_rdenS(s t) => BUSN(R_srS(s t)) | Offn),Offn))"),
    REWRITE_TAC [SYM_RULE ONE_LESS_EQ]
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC (SYM_RULE (SPECL ["t:time";"1"] SUB_ADD))
    THEN ONCE_ASM_REWRITE_TAC[]
    THEN UNDISCH_TAC "RBlock_GATE rep s e p"
    THEN POP_ASSUM_LIST (MAP_EVERY (\th. ALL_TAC))
    THEN REWRITE_TAC [RBlock_EXP;(EXPAND_LET_RULE Bus12n_CF);ASel;BSel]
    THEN REPEAT STRIP_TAC
    THEN POP_ASSUM_LIST (MAP_EVERY (\th. ASSUME_TAC (SPEC "t-1" th)))
    THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "0<=31"))
    THEN IMP_RES_TAC (SPECL ["31";"0"] OFFnP_Offn)
    THEN ASM_REWRITE_TAC [lemma1;lemma2]
    THEN POP_ASSUM_LIST (MAP_EVERY (\th. ALL_TAC))
    THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "0<=31"))
    THEN IMP_RES_TAC (SPECL ["31";"0"] OFFnP_Offn)
    THEN IMP_RES_TAC (SPECL ["31";"0"] ONnP_Offn)
    THEN IMP_RES_TAC (GEN_ALL(SPECL ["f:wordn";"31";"0"] OFFnP_BUSN))
    THEN ASM_CASES_TAC (EXPANSION_EQ "WORDN 8")
    THEN ASM_REWRITE_TAC ([wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN] @
                          (WORDN_NOT_EQ_THMLIST "8"))
    THEN ASM_CASES_TAC (EXPANSION_EQ "WORDN 12")
    THEN ASM_REWRITE_TAC ([wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN] @
                          (WORDN_NOT_EQ_THMLIST "12"))
    THEN ASM_CASES_TAC (EXPANSION_EQ "WORDN 9")
    THEN ASM_REWRITE_TAC ([wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN] @
                          (WORDN_NOT_EQ_THMLIST "9"))
    THEN ASM_CASES_TAC (EXPANSION_EQ "WORDN 13")
    THEN ASM_REWRITE_TAC ([wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN] @
                          (WORDN_NOT_EQ_THMLIST "13"))
    THEN ASM_CASES_TAC (EXPANSION_EQ "WORDN 10")
    THEN ASM_REWRITE_TAC ([wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN] @
                          (WORDN_NOT_EQ_THMLIST "10"))
    THEN ASM_CASES_TAC (EXPANSION_EQ "WORDN 14")
    THEN ASM_REWRITE_TAC ([wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN] @
                          (WORDN_NOT_EQ_THMLIST "14"))
    THEN ASM_CASES_TAC (EXPANSION_EQ "WORDN 11")
    THEN ASM_REWRITE_TAC ([wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN] @
                          (WORDN_NOT_EQ_THMLIST "11"))
    THEN ASM_CASES_TAC (EXPANSION_EQ "WORDN 15")
```

```
      THEN ASM_REWRITE_TAC ([wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN] @
                            (WORDN_NOT_EQ_THMLIST "15"))
      THEN ASM_CASES_TAC (EXPANSION_EQ "WORDN 3")
      THEN ASM_REWRITE_TAC ([wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN] @
                            (WORDN_NOT_EQ_THMLIST "3"))
      THEN ASM_CASES_TAC (EXPANSION_EQ "WORDN 2")
      THEN ASM_REWRITE_TAC ([wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN] @
                            (WORDN_NOT_EQ_THMLIST "2"))
      THEN ASM_CASES_TAC (EXPANSION_EQ "WORDN 4")
      THEN ASM_REWRITE_TAC ([wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN] @
                            (WORDN_NOT_EQ_THMLIST "4"))
      THEN REWRITE_TAC [COND_TRUE_CHOICES]
     );;

set_flag ('print_all_subgoals', false);;

let R_ctr0S_THM = prove_thm
     ('R_ctr0S',
      "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
         (p :time->r_out) .
         RBlock_GATE rep s e p
            ==>
         (R_ctr0S (RClockNSF rep (s t) (e t)) = R_ctr0S (s (t+1)))",
      REWRITE_TAC [R_ctr0S;RBlock_EXP;RClockNSF_EXP]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let R_ctr1S_THM = prove_thm
     ('R_ctr1S',
      "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
         (p :time->r_out) .
         RBlock_GATE rep s e p
            ==>
         (R_ctr1S (RClockNSF rep (s t) (e t)) = R_ctr1S (s (t+1)))",
      REWRITE_TAC [R_ctr1S;RBlock_EXP;RClockNSF_EXP]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let R_ctr2S_THM = prove_thm
     ('R_ctr2S',
      "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
         (p :time->r_out) .
         RBlock_GATE rep s e p
            ==>
         (R_ctr2S (RClockNSF rep (s t) (e t)) = R_ctr2S (s (t+1)))",
      REWRITE_TAC [R_ctr2S;RBlock_EXP;RClockNSF_EXP]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let R_ctr3S_THM = prove_thm
     ('R_ctr3S',
      "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
         (p :time->r_out) .
         RBlock_GATE rep s e p
            ==>
         (R_ctr3S (RClockNSF rep (s t) (e t)) = R_ctr3S (s (t+1)))",
      REWRITE_TAC [R_ctr3S;RBlock_EXP;RClockNSF_EXP]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]);;
     );;

let R_busA_latchS_THM = prove_thm
     ('R_busA_latchS',
      "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
         (p :time->r_out) .
         RBlock_GATE rep s e p /\ t > 0
            ==>
         (R_busA_latchS (RClockNSF rep (s t) (e t)) = R_busA_latchS (s (t+1)))",
      REPEAT STRIP_TAC
```

```
    THEN IMP_RES_TAC lemmaCF
    THEN UNDISCH_TAC "RBlock_GATE rep s e p"
    THEN REWRITE_TAC [R_busA_latchS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
    THEN POP_ASSUM_LIST (MAP_EVERY (\th. ALL_TAC))
    THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "0<=31"))
    THEN IMP_RES_TAC (SPECL ["31";"0"] OFFnP_Offn)
    THEN IMP_RES_TAC (SPECL ["31";"0"] ONnP_Offn)
    THEN IMP_RES_TAC (GEN_ALL(SPECL ["f:wordn";"31";"0"] OFFnP_BUSN))
    THEN ASM_CASES_TAC "R_ctr0_irdenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_ctr0_ordenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_ctr1_irdenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_ctr1_ordenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_ctr2_irdenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_ctr2_ordenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_ctr3_irdenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_ctr3_ordenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_icr_rdenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_ccr_rdenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_gcr_rdenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_sr_rdenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN UNDISCH_TAC "RBlock_GATE rep s e p"
    THEN POP_ASSUM_LIST (MAP_EVERY (\th. ALL_TAC))
    THEN REWRITE_TAC [RBlock_EXP;(EXPAND_LET_RULE Bus12n_CF);ASel;BSel]
    THEN REPEAT STRIP_TAC
    THEN POP_ASSUM_LIST (MAP_EVERY (\th. ASSUME_TAC (SPEC "t-1" th)))
    THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "0<=31"))
    THEN IMP_RES_TAC (SPECL ["31";"0"] OFFnP_Offn)
    THEN ASM_REWRITE_TAC [lemma1;lemma2]
  );;

let R_fsm_stateS_THM = prove_thm
   ('R_fsm_stateS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_fsm_stateS (RClockNSF rep (s t) (e t)) = R_fsm_stateS (s (t+1)))",
    REWRITE_TAC [R_fsm_stateS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let R_fsm_ale_S_THM = prove_thm
   ('R_fsm_ale_S',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_fsm_ale_S (RClockNSF rep (s t) (e t)) = R_fsm_ale_S (s (t+1)))",
    REWRITE_TAC [R_fsm_ale_S;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let R_fsm_mrdy_S_THM = prove_thm
   ('R_fsm_mrdy_S',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
```

```
         RBlock_GATE rep s e p
            ==>
         (R_fsm_mrdy_S (RClockNSF rep (s t) (e t)) = R_fsm_mrdy_S (s (t+1)))",
      REWRITE_TAC [R_fsm_mrdy_S;RBlock_EXP;RClockNSF_EXP]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let R_fsm_last_S_THM = prove_thm
     ('R_fsm_last_S',
      "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
         (p :time->r_out) .
         RBlock_GATE rep s e p
            ==>
         (R_fsm_last_S (RClockNSF rep (s t) (e t)) = R_fsm_last_S (s (t+1)))",
      REWRITE_TAC [R_fsm_last_S;RBlock_EXP;RClockNSF_EXP]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let R_fsm_rstS_THM = prove_thm
     ('R_fsm_rstS',
      "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
         (p :time->r_out) .
         RBlock_GATE rep s e p
            ==>
         (R_fsm_rstS (RClockNSF rep (s t) (e t)) = R_fsm_rstS (s (t+1)))",
      REWRITE_TAC [R_fsm_rstS;RBlock_EXP;RClockNSF_EXP]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let R_int0_disS_THM = prove_thm
     ('R_int0_disS',
      "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
         (p :time->r_out) .
         RBlock_GATE rep s e p
            ==>
         (R_int0_disS (RClockNSF rep (s t) (e t)) = R_int0_disS (s (t+1)))",
      REWRITE_TAC [R_int0_disS;RBlock_EXP;RClockNSF_EXP]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let R_int3_disS_THM = prove_thm
     ('R_int3_disS',
      "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
         (p :time->r_out) .
         RBlock_GATE rep s e p
            ==>
         (R_int3_disS (RClockNSF rep (s t) (e t)) = R_int3_disS (s (t+1)))",
      REWRITE_TAC [R_int3_disS;RBlock_EXP;RClockNSF_EXP]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let R_c01_cout_delS = prove_thm
     ('R_c01_cout_delS',
      "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
         (p :time->r_out) .
         RBlock_GATE rep s e p
            ==>
         (R_c01_cout_delS (RClockNSF rep (s t) (e t)) =
            R_c01_cout_delS (s (t+1)))",
      REWRITE_TAC [R_c01_cout_delS;RBlock_EXP;RClockNSF_EXP]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let R_int1_enS_THM = prove_thm
     ('R_int1_enS',
      "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
      (p :time->r_out) .
      RBlock_GATE rep s e p
         ==>
      (R_int1_enS (RClockNSF rep (s t) (e t)) = R_int1_enS (s (t+1)))",
   REWRITE_TAC [R_int1_enS;RBlock_EXP;RClockNSF_EXP]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC
          [SPECL
            ["((R_fsm_rstS((s:time->r_state) t) => RI |
              ((R_fsm_stateS(s t) = RI) =>
              ((~R_fsm_ale_S(s t)) => RA | RI) |
              ((R_fsm_stateS(s t) = RA) =>
              ((~R_fsm_mrdy_S(s t)) => RD | RA) |
              ((~R_fsm_last_S(s t)) => RI | RA)))) = RD)";
             "((R_fsm_rstS((s:time->r_state) t) => RI |
              ((R_fsm_stateS(s t) = RI) =>
              ((~R_fsm_ale_S(s t)) => RA | RI) |
              ((R_fsm_stateS(s t) = RA) =>
              ((~R_fsm_mrdy_S(s t)) => RD | RA) |
              ((~R_fsm_last_S(s t)) => RI | RA)))) = RA)"]
            lemma1]
  );;

let R_c23_cout_delS_THM = prove_thm
  ('R_c23_cout_delS',
   "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
      (p :time->r_out) .
      RBlock_GATE rep s e p
         ==>
      (R_c23_cout_delS (RClockNSF rep (s t) (e t)) =
          R_c23_cout_delS (s (t+1)))",
   REWRITE_TAC [R_c23_cout_delS;RBlock_EXP;RClockNSF_EXP]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC[]
  );;

let R_int2_enS_THM = prove_thm
  ('R_int2_enS',
   "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
      (p :time->r_out) .
      RBlock_GATE rep s e p
         ==>
      (R_int2_enS (RClockNSF rep (s t) (e t)) = R_int2_enS (s (t+1)))",
   REWRITE_TAC [R_int2_enS;RBlock_EXP;RClockNSF_EXP]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC[lemma1_spec]
  );;

let R_wrS_THM = prove_thm
  ('R_wrS',
   "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
      (p :time->r_out) .
      RBlock_GATE rep s e p
         ==>
      (R_wrS (RClockNSF rep (s t) (e t)) = R_wrS (s (t+1)))",
   REWRITE_TAC [R_wrS;RBlock_EXP;RClockNSF_EXP]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC[]
  );;

let R_cntlatch_delS_THM = prove_thm
  ('R_cntlatch_delS',
   "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
      (p :time->r_out) .
      RBlock_GATE rep s e p
         ==>
      (R_cntlatch_delS (RClockNSF rep (s t) (e t)) =
         R_cntlatch_delS (s (t+1)))",
   REWRITE_TAC [R_cntlatch_delS;RBlock_EXP;RClockNSF_EXP]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC[]
  );;
```

```
let R_srdy_del_S_THM = prove_thm
   ('R_srdy_del_S',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_srdy_del_S (RClockNSF rep (s t) (e t)) = R_srdy_del_S (s (t+1)))",
    REWRITE_TAC [R_srdy_del_S;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let R_reg_selS_THM = prove_thm
   ('R_reg_selS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_reg_selS (RClockNSF rep (s t) (e t)) = R_reg_selS (s (t+1)))",
    REWRITE_TAC [R_reg_selS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let R_ctr0_inS_THM = prove_thm
   ('R_ctr0_inS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr0_inS (RClockNSF rep (s t) (e t)) = R_ctr0_inS (s (t+1)))",
    REWRITE_TAC [R_ctr0_inS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec]
   );;

let R_ctr0_mux_selS_THM = prove_thm
   ('R_ctr0_mux_selS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr0_mux_selS (RClockNSF rep (s t) (e t)) =
           R_ctr0_mux_selS (s (t+1)))",
    REWRITE_TAC [R_ctr0_mux_selS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec]
   );;

let R_ctr0_irdenS_THM = prove_thm
   ('R_ctr0_irdenS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr0_irdenS (RClockNSF rep (s t) (e t)) = R_ctr0_irdenS (s (t+1)))",
    REWRITE_TAC [R_ctr0_irdenS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr0_cryS_THM = prove_thm
   ('R_ctr0_cryS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr0_cryS (RClockNSF rep (s t) (e t)) = R_ctr0_cryS (s (t+1)))",
    REWRITE_TAC [R_ctr0_cryS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr0_newS_THM = prove_thm
   ('R_ctr0_newS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr0_newS (RClockNSF rep (s t) (e t)) = R_ctr0_newS (s (t+1)))",
    REWRITE_TAC [R_ctr0_newS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr0_outS_THM = prove_thm
   ('R_ctr0_outS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr0_outS (RClockNSF rep (s t) (e t)) = R_ctr0_outS (s (t+1)))",
    REWRITE_TAC [R_ctr0_outS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr0_ordenS_THM = prove_thm
   ('R_ctr0_ordenS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr0_ordenS (RClockNSF rep (s t) (e t)) = R_ctr0_ordenS (s (t+1)))",
    REWRITE_TAC [R_ctr0_ordenS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr1_inS_THM = prove_thm
   ('R_ctr1_inS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr1_inS (RClockNSF rep (s t) (e t)) = R_ctr1_inS (s (t+1)))",
    REWRITE_TAC [R_ctr1_inS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr1_mux_selS_THM = prove_thm
   ('R_ctr1_mux_selS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr1_mux_selS (RClockNSF rep (s t) (e t)) =
            R_ctr1_mux_selS (s (t+1)))",
    REWRITE_TAC [R_ctr1_mux_selS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr1_irdenS_THM = prove_thm
   ('R_ctr1_irdenS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr1_irdenS (RClockNSF rep (s t) (e t)) = R_ctr1_irdenS (s (t+1)))",
    REWRITE_TAC [R_ctr1_irdenS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctrl_cryS_THM = prove_thm
   ('R_ctrl_cryS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctrl_cryS (RClockNSF rep (s t) (e t)) = R_ctrl_cryS (s (t+1)))",
    REWRITE_TAC [R_ctrl_cryS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctrl_newS_THM = prove_thm
   ('R_ctrl_newS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctrl_newS (RClockNSF rep (s t) (e t)) = R_ctrl_newS (s (t+1)))",
    REWRITE_TAC [R_ctrl_newS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctrl_outS_THM = prove_thm
   ('R_ctrl_outS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctrl_outS (RClockNSF rep (s t) (e t)) = R_ctrl_outS (s (t+1)))",
    REWRITE_TAC [R_ctrl_outS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctrl_ordenS_THM = prove_thm
   ('R_ctrl_ordenS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctrl_ordenS (RClockNSF rep (s t) (e t)) = R_ctrl_ordenS (s (t+1)))",
    REWRITE_TAC [R_ctrl_ordenS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr2_inS_THM = prove_thm
   ('R_ctr2_inS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr2_inS (RClockNSF rep (s t) (e t)) = R_ctr2_inS (s (t+1)))",
    REWRITE_TAC [R_ctr2_inS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr2_mux_selS_THM = prove_thm
   ('R_ctr2_mux_selS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr2_mux_selS (RClockNSF rep (s t) (e t)) =
           R_ctr2_mux_selS (s (t+1)))",
    REWRITE_TAC [R_ctr2_mux_selS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr2_irdenS_THM = prove_thm
   ('R_ctr2_irdenS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr2_irdenS (RClockNSF rep (s t) (e t)) = R_ctr2_irdenS (s (t+1)))",
    REWRITE_TAC [R_ctr2_irdenS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr2_cryS_THM = prove_thm
   ('R_ctr2_cryS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr2_cryS (RClockNSF rep (s t) (e t)) = R_ctr2_cryS (s (t+1)))",
    REWRITE_TAC [R_ctr2_cryS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr2_newS_THM = prove_thm
   ('R_ctr2_newS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr2_newS (RClockNSF rep (s t) (e t)) = R_ctr2_newS (s (t+1)))",
    REWRITE_TAC [R_ctr2_newS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr2_outS_THM = prove_thm
   ('R_ctr2_outS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr2_outS (RClockNSF rep (s t) (e t)) = R_ctr2_outS (s (t+1)))",
    REWRITE_TAC [R_ctr2_outS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr2_ordenS_THM = prove_thm
   ('R_ctr2_ordenS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr2_ordenS (RClockNSF rep (s t) (e t)) = R_ctr2_ordenS (s (t+1)))",
    REWRITE_TAC [R_ctr2_ordenS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr3_inS_THM = prove_thm
   ('R_ctr3_inS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr3_inS (RClockNSF rep (s t) (e t)) = R_ctr3_inS (s (t+1)))",
    REWRITE_TAC [R_ctr3_inS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr3_mux_selS_THM = prove_thm
   ('R_ctr3_mux_selS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr3_mux_selS (RClockNSF rep (s t) (e t)) =
           R_ctr3_mux_selS (s (t+1)))",
    REWRITE_TAC [R_ctr3_mux_selS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr3_irdenS_THM = prove_thm
   ('R_ctr3_irdenS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr3_irdenS (RClockNSF rep (s t) (e t)) = R_ctr3_irdenS (s (t+1)))",
    REWRITE_TAC [R_ctr3_irdenS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr3_cryS_THM = prove_thm
   ('R_ctr3_cryS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr3_cryS (RClockNSF rep (s t) (e t)) = R_ctr3_cryS (s (t+1)))",
    REWRITE_TAC [R_ctr3_cryS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr3_newS_THM = prove_thm
   ('R_ctr3_newS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr3_newS (RClockNSF rep (s t) (e t)) = R_ctr3_newS (s (t+1)))",
    REWRITE_TAC [R_ctr3_newS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr3_outS_THM = prove_thm
   ('R_ctr3_outS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr3_outS (RClockNSF rep (s t) (e t)) = R_ctr3_outS (s (t+1)))",
    REWRITE_TAC [R_ctr3_outS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ctr3_ordenS_THM = prove_thm
   ('R_ctr3_ordenS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ctr3_ordenS (RClockNSF rep (s t) (e t)) = R_ctr3_ordenS (s (t+1)))",
    REWRITE_TAC [R_ctr3_ordenS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_icr_loadS_THM = prove_thm
   ('R_icr_loadS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_icr_loadS (RClockNSF rep (s t) (e t)) = R_icr_loadS (s (t+1)))",
    REWRITE_TAC [R_icr_loadS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_icr_oldS_THM = prove_thm
   ('R_icr_oldS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_icr_oldS (RClockNSF rep (s t) (e t)) = R_icr_oldS (s (t+1)))",
    REWRITE_TAC [R_icr_oldS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_icr_maskS_THM = prove_thm
   ('R_icr_maskS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_icr_maskS (RClockNSF rep (s t) (e t)) = R_icr_maskS (s (t+1)))",
    REWRITE_TAC [R_icr_maskS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_icrS_THM = prove_thm
   ('R_icrS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_icrS (RClockNSF rep (s t) (e t)) = R_icrS (s (t+1)))",
    REWRITE_TAC [R_icrS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_icr_rdenS_THM = prove_thm
   ('R_icr_rdenS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_icr_rdenS (RClockNSF rep (s t) (e t)) = R_icr_rdenS (s (t+1)))",
    REWRITE_TAC [R_icr_rdenS;RBlock_EXP;RClockNSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
   );;

let R_ccrS_THM = prove_thm
   ('R_ccrS',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (R_ccrS (RClockNSF rep (s t) (e t)) = R_ccrS (s (t+1)))",
        REWRITE_TAC [R_ccrS;RBlock_EXP;RClockNSF_EXP]
        THEN REPEAT STRIP_TAC
        THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
       );;

    let R_ccr_rdenS_THM = prove_thm
       ('R_ccr_rdenS',
        "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
           (p :time->r_out) .
           RBlock_GATE rep s e p
              ==>
           (R_ccr_rdenS (RClockNSF rep (s t) (e t)) = R_ccr_rdenS (s (t+1)))",
        REWRITE_TAC [R_ccr_rdenS;RBlock_EXP;RClockNSF_EXP]
        THEN REPEAT STRIP_TAC
        THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
       );;

    let R_gcrS_THM = prove_thm
       ('R_gcrS',
        "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
           (p :time->r_out) .
           RBlock_GATE rep s e p
              ==>
           (R_gcrS (RClockNSF rep (s t) (e t)) = R_gcrS (s (t+1)))",
        REWRITE_TAC [R_gcrS;RBlock_EXP;RClockNSF_EXP]
        THEN REPEAT STRIP_TAC
        THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
       );;

    let R_gcr_rdenS_THM = prove_thm
       ('R_gcr_rdenS',
        "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
           (p :time->r_out) .
           RBlock_GATE rep s e p
              ==>
           (R_gcr_rdenS (RClockNSF rep (s t) (e t)) = R_gcr_rdenS (s (t+1)))",
        REWRITE_TAC [R_gcr_rdenS;RBlock_EXP;RClockNSF_EXP]
        THEN REPEAT STRIP_TAC
        THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
       );;

    let R_srS_THM = prove_thm
       ('R_srS',
        "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
           (p :time->r_out) .
           RBlock_GATE rep s e p
              ==>
           (R_srS (RClockNSF rep (s t) (e t)) = R_srS (s (t+1)))",
        REWRITE_TAC [R_srS;RBlock_EXP;RClockNSF_EXP]
        THEN REPEAT STRIP_TAC
        THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
       );;

    let R_sr_rdenS_THM = prove_thm
       ('R_sr_rdenS',
        "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
           (p :time->r_out) .
           RBlock_GATE rep s e p
              ==>
           (R_sr_rdenS (RClockNSF rep (s t) (e t)) = R_sr_rdenS (s (t+1)))",
        REWRITE_TAC [R_sr_rdenS;RBlock_EXP;RClockNSF_EXP]
        THEN REPEAT STRIP_TAC
        THEN ASM_REWRITE_TAC[lemma1_spec;lemma2_spec]
       );;

    let I_ad_outO_THM = prove_thm
       ('I_ad_outO',
        "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
           (p :time->r_out) .
           RBlock_GATE rep s e p /\ t > 0
              ==>
           (I_ad_outO (RClockOF rep (s t) (e t)) = I_ad_outO (p t))",
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC lemmaCF
    THEN UNDISCH_TAC "RBlock_GATE rep s e p"
    THEN REWRITE_TAC [I_ad_out0;RBlock_EXP;RClockOF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1;lemma2;lemma3]
    THEN POP_ASSUM_LIST (MAP_EVERY (\th. ALL_TAC))
    THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "0<=31"))
    THEN IMP_RES_TAC (SPECL ["31";"0"] OFFnP_Offn)
    THEN IMP_RES_TAC (SPECL ["31";"0"] ONnP_Offn)
    THEN IMP_RES_TAC (GEN_ALL(SPECL ["f:wordn";"31";"0"] OFFnP_BUSN))
    THEN ASM_CASES_TAC "R_ctr0_irdenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_ctr0_ordenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_ctr1_irdenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_ctr1_ordenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_ctr2_irdenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_ctr2_ordenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_ctr3_irdenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_ctr3_ordenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_icr_rdenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_ccr_rdenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_gcr_rdenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
    THEN ASM_CASES_TAC "R_sr_rdenS((s:time->r_state) t)"
    THEN ASM_REWRITE_TAC [wordnVAL_BUSN_IDENT;wordnVAL_Offn;ONnP_BUSN]
  );;

let I_srdy_O_THM = prove_thm
   ('I_srdy_O',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (I_srdy_O (RClockOF rep (s t) (e t)) = I_srdy_O (p t))",
    REWRITE_TAC [I_srdy_O;RBlock_EXP;RClockOF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1;lemma2;lemma3]
   );;

let Int0_O_THM = prove_thm
   ('Int0_O',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (Int0_O (RClockOF rep (s t) (e t)) = Int0_O (p t))",
    REWRITE_TAC [Int0_O;RBlock_EXP;RClockOF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1;lemma2]
   );;

let Int1O_THM = prove_thm
   ('Int1O',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (Int1O (RClockOF rep (s t) (e t)) = Int1O (p t))",
    REWRITE_TAC [Int1O;RBlock_EXP;RClockOF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1;lemma2]
   );;
```

```
let Int2O_THM = prove_thm
   ('Int2O',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (Int2O (RClockOF rep (s t) (e t)) = Int2O (p t))",
    REWRITE_TAC [Int2O;RBlock_EXP;RClockOF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1;lemma2]
   );;

let Int3_O_THM = prove_thm
   ('Int3_O',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (Int3_O (RClockOF rep (s t) (e t)) = Int3_O (p t))",
    REWRITE_TAC [Int3_O;RBlock_EXP;RClockOF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1;lemma2]
   );;

let CcrO_THM = prove_thm
   ('CcrO',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (CcrO (RClockOF rep (s t) (e t)) = CcrO (p t))",
    REWRITE_TAC [CcrO;RBlock_EXP;RClockOF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1;lemma2]
   );;

let LedO_THM = prove_thm
   ('LedO',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (LedO (RClockOF rep (s t) (e t)) = LedO (p t))",
    REWRITE_TAC [LedO;RBlock_EXP;RClockOF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1;lemma2]
   );;

let Reset_errorO_THM = prove_thm
   ('Reset_errorO',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (Reset_errorO (RClockOF rep (s t) (e t)) = Reset_errorO (p t))",
    REWRITE_TAC [Reset_errorO;RBlock_EXP;RClockOF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1;lemma2]
   );;

let Pmm_invalidO_THM = prove_thm
   ('Pmm_invalidO',
    "! (t :time) (rep :^REP_ty) (s :time->r_state) (e :time->r_env)
       (p :time->r_out) .
       RBlock_GATE rep s e p
          ==>
       (Pmm_invalidO (RClockOF rep (s t) (e t)) = Pmm_invalidO (p t))",
    REWRITE_TAC [Pmm_invalidO;RBlock_EXP;RClockOF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[lemma1;lemma2]
   );;
```

3.4 C-Port Clock-Level Verification

The theory *cclock_ver* and file *cc_thms.ml* contain the C-Port clock-level correctness proof.

```
%-------------------------------------------------------------------------------

   File:         cclock_ver.ml

   Author:       (c) D.A. Fura 1992-93

   Date:         4 March 1993

-------------------------------------------------------------------------------%

set_flag ('timing', true);;

set_search_path (search_path() @ ['/home/elvis6/dfura/ftep/piu/hol/cport/';
                                  '/home/elvis6/dfura/ftep/piu/hol/lib/';
                                  '/home/elvis6/dfura/hol/ml/';
                                  '/home/elvis6/dfura/hol/Library/abs_theory/';
                                  '/home/elvis6/dfura/hol/Library/time/';
                                  '/home/elvis6/dfura/hol/Library/GI/';
                                  '/home/elvis6/dfura/hol/Library/tools/'
                                 ]);;
system 'rm cclock_ver.th';;

new_theory 'cclock_ver';;

loadf 'abs_theory';;

loadf 'aux_defs';;

map new_parent ['piuaux_def';'wordn_def';'array_def';
                'cfsms_def';'gates_def1';'latches_def';'ffs_def';
                'counters_def';'ineq'];;

map load_parent ['caux_def';'cblock_def';'cclock_def'];;

load_parent 'time_abs';;

let REP_ty = abs_type_info (theorem 'piuaux_def' 'REP');;

let CC_OF_REW = theorem 'cclock_def' 'CC_OF_REW';;
let CBlock_EXP = theorem 'cblock_def' 'CBlock_EXP';;

let MSTART = "WORDN 2 4";;
let MEND = "WORDN 2 5";;
let MRDY = "WORDN 2 6";;
let MWAIT = "WORDN 2 7";;
let MABORT = "WORDN 2 0";;

let SACK = "WORDN 2 5";;
let SRDY = "WORDN 2 6";;
let SWAIT = "WORDN 2 7";;
let SABORT = "WORDN 2 0";;

let CC_NSF_EXP = EXPAND_LET_RULE CC_NSF_REW;;
let CC_OF_EXP = EXPAND_LET_RULE CC_OF_REW;;

loadt 'cc_thms.ml';;

let C_Clock_Correct = prove_thm
   ('C_Clock_Correct',
    "! (t :time) (s :time->cc_state) (e :time->cc_env) (p :time->cc_out) .
       CBlock_GATE rep s e p
```

```
          ==>
       CCSet_Correct rep s e p",
    REPEAT STRIP_TAC
    THEN REWRITE_TAC [CCSet_Correct]
    THEN INDUCT_THEN (prove_induction_thm CCI) ASSUME_TAC
    THEN GEN_TAC
    THEN REWRITE_TAC [CC_Correct;CC_Exec;CC_PreC;CC_PostC]
    THEN CONJ_TAC
    THENL [
       % Subgoal 1: "s(t + 1) = CC_NSF(s t)(e t)" %
       SUBST_TAC [SPEC "(s (t+1)):cc_state" CCState_Selectors_Work]
       THEN IMP_RES_TAC (SYM_RULE C_mfsm_state_THM)
       THEN IMP_RES_TAC (SYM_RULE C_mfsm_srdy_en_THM)
       THEN IMP_RES_TAC (SYM_RULE C_mfsm_D_THM)
       THEN IMP_RES_TAC (SYM_RULE C_mfsm_grant_THM)
       THEN IMP_RES_TAC (SYM_RULE C_mfsm_rst_THM)
       THEN IMP_RES_TAC (SYM_RULE C_mfsm_busy_THM)
       THEN IMP_RES_TAC (SYM_RULE C_mfsm_write_THM)
       THEN IMP_RES_TAC (SYM_RULE C_mfsm_crqt_THM)
       THEN IMP_RES_TAC (SYM_RULE C_mfsm_hold_THM)
       THEN IMP_RES_TAC (SYM_RULE C_mfsm_last_THM)
       THEN IMP_RES_TAC (SYM_RULE C_mfsm_lock_THM)
       THEN IMP_RES_TAC (SYM_RULE C_mfsm_lock_THM)
       THEN IMP_RES_TAC (SYM_RULE C_mfsm_invalid_THM)
       THEN IMP_RES_TAC (SYM_RULE C_sfsm_state_THM)
       THEN IMP_RES_TAC (SYM_RULE C_sfsm_D_THM)
       THEN IMP_RES_TAC (SYM_RULE C_sfsm_grant_THM)
       THEN IMP_RES_TAC (SYM_RULE C_sfsm_rst_THM)
       THEN IMP_RES_TAC (SYM_RULE C_sfsm_write_THM)
       THEN IMP_RES_TAC (SYM_RULE C_sfsm_addressed_THM)
       THEN IMP_RES_TAC (SYM_RULE C_sfsm_hlda_THM)
       THEN IMP_RES_TAC (SYM_RULE C_sfsm_ms_THM)
       THEN IMP_RES_TAC (SYM_RULE C_efsm_state_THM)
       THEN IMP_RES_TAC (SYM_RULE C_efsm_cale_THM)
       THEN IMP_RES_TAC (SYM_RULE C_efsm_last_THM)
       THEN IMP_RES_TAC (SYM_RULE C_efsm_male_THM)
       THEN IMP_RES_TAC (SYM_RULE C_efsm_rale_THM)
       THEN IMP_RES_TAC (SYM_RULE C_efsm_srdy_THM)
       THEN IMP_RES_TAC (SYM_RULE C_efsm_rst_THM)
       THEN IMP_RES_TAC (SYM_RULE C_lock_in_THM)
       THEN IMP_RES_TAC (SYM_RULE C_last_in_THM)
       THEN IMP_RES_TAC (SYM_RULE C_ss_THM)
       THEN IMP_RES_TAC (SYM_RULE C_clkA_THM)
       THEN IMP_RES_TAC (SYM_RULE C_last_out_THM)
       THEN IMP_RES_TAC (SYM_RULE C_sidle_del_THM)
       THEN IMP_RES_TAC (SYM_RULE C_mrqt_del_THM)
       THEN IMP_RES_TAC (SYM_RULE C_hold_THM)
       THEN IMP_RES_TAC (SYM_RULE C_cout_0_le_del_THM)
       THEN IMP_RES_TAC (SYM_RULE C_cin_2_le_THM)
       THEN IMP_RES_TAC (SYM_RULE C_mrdy_del_THM)
       THEN IMP_RES_TAC (SYM_RULE C_iad_en_s_del_THM)
       THEN IMP_RES_TAC (SYM_RULE C_wrdy_THM)
       THEN IMP_RES_TAC (SYM_RULE C_rrdy_THM)
       THEN IMP_RES_TAC (SYM_RULE C_parity_THM)
       THEN IMP_RES_TAC (SYM_RULE C_source_THM)
       THEN IMP_RES_TAC (SYM_RULE C_data_in_THM)
       THEN IMP_RES_TAC (SYM_RULE C_sizewrbe_THM)
       THEN IMP_RES_TAC (SYM_RULE C_iad_out_THM)
       THEN IMP_RES_TAC (SYM_RULE C_a1a0_THM)
       THEN IMP_RES_TAC (SYM_RULE C_a3a2_THM)
       THEN IMP_RES_TAC (SYM_RULE C_iad_in_THM)
       THEN IMP_RES_TAC (SYM_RULE C_wr_THM)
       THEN ASM_REWRITE_TAC
             [SPEC "CC_NSF (rep:^REP_ty) ((s:time->cc_state) t)
                           ((e:time->cc_env) t)"
                   (SYM_RULE CCState_Selectors_Work)]
    ;
       % Subgoal 2: "p t = CC_OF(s t)(e t)" %
       SUBST_TAC [SPEC "((p:time->cc_out) t)" CCOut_Selectors_Work]
       THEN IMP_RES_TAC (SYM_RULE I_cgnt_THM)
       THEN IMP_RES_TAC (SYM_RULE I_mrdy_out_THM)
       THEN IMP_RES_TAC (SYM_RULE I_hold_THM)
```

```
        THEN IMP_RES_TAC (SYM_RULE I_rale_out_THM)
        THEN IMP_RES_TAC (SYM_RULE I_male_out_THM)
        THEN IMP_RES_TAC (SYM_RULE I_last_out_THM)
        THEN IMP_RES_TAC (SYM_RULE I_srdy_out_THM)
        THEN IMP_RES_TAC (SYM_RULE I_ad_out_THM)
        THEN IMP_RES_TAC (SYM_RULE I_be_out_THM)
        THEN IMP_RES_TAC (SYM_RULE CB_rqt_out_THM)
        THEN IMP_RES_TAC (SYM_RULE CB_ms_out_THM)
        THEN IMP_RES_TAC (SYM_RULE CB_ss_out_THM)
        THEN IMP_RES_TAC (SYM_RULE CB_ad_out_THM)
        THEN IMP_RES_TAC (SYM_RULE C_ss_out_THM)
        THEN IMP_RES_TAC (SYM_RULE Disable_writes_THM)
        THEN IMP_RES_TAC (SYM_RULE CB_parity_THM)
        THEN ASM_REWRITE_TAC
              [SPEC "CC_OF (rep:^REP_ty) ((s:time->cc_state) t)
                          ((e:time->cc_env) t)"
                    (SYM_RULE CCOut_Selectors_Work)]
     ]
   );;

close_theory();;


%---------------------------------------------------------------------------

   File:          cc_thms.ml

   Author:        (c) D.A. Fura 1992-93

   Date:          3 March 1993

---------------------------------------------------------------------------%

let C_mfsm_stateS_THM = TAC_PROOF
   (([],
    "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
       (p :time->cc_out) .
       CBlock_GATE rep s e p
          ==>
       (C_mfsm_stateS (CC_NSF rep (s t) (e t)) = C_mfsm_stateS (s (t+1)))"),
    REWRITE_TAC [C_mfsm_stateS;CBlock_EXP;CC_NSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let C_mfsm_srdy_enS_THM = TAC_PROOF
   (([],
    "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
       (p :time->cc_out) .
       CBlock_GATE rep s e p
          ==>
       (C_mfsm_srdy_enS (CC_NSF rep (s t) (e t)) = C_mfsm_srdy_enS (s (t+1)))"),
    REWRITE_TAC [C_mfsm_srdy_enS;CBlock_EXP;CC_NSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let C_mfsm_DS_THM = TAC_PROOF
   (([],
    "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
       (p :time->cc_out) .
       CBlock_GATE rep s e p
          ==>
       (C_mfsm_DS (CC_NSF rep (s t) (e t)) = C_mfsm_DS (s (t+1)))"),
    REWRITE_TAC [C_mfsm_DS;CBlock_EXP;CC_NSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let C_mfsm_grantS_THM = TAC_PROOF
   (([],
    "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
```

```
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_mfsm_grantS (CC_NSF rep (s t) (e t)) = C_mfsm_grantS (s (t+1)))"),
     REWRITE_TAC [C_mfsm_grantS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_mfsm_rstS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_mfsm_rstS (CC_NSF rep (s t) (e t)) = C_mfsm_rstS (s (t+1)))"),
     REWRITE_TAC [C_mfsm_rstS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_mfsm_busyS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_mfsm_busyS (CC_NSF rep (s t) (e t)) = C_mfsm_busyS (s (t+1)))"),
     REWRITE_TAC [C_mfsm_busyS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_mfsm_writeS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_mfsm_writeS (CC_NSF rep (s t) (e t)) = C_mfsm_writeS (s (t+1)))"),
     REWRITE_TAC [C_mfsm_writeS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_mfsm_crqt_S_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_mfsm_crqt_S (CC_NSF rep (s t) (e t)) = C_mfsm_crqt_S (s (t+1)))"),
     REWRITE_TAC [C_mfsm_crqt_S;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_mfsm_hold_S_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_mfsm_hold_S (CC_NSF rep (s t) (e t)) = C_mfsm_hold_S (s (t+1)))"),
     REWRITE_TAC [C_mfsm_hold_S;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_mfsm_last_S_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
```

```
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_mfsm_last_S (CC_NSF rep (s t) (e t)) = C_mfsm_last_S (s (t+1)))"),
     REWRITE_TAC [C_mfsm_last_S;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_mfsm_lock_S_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_mfsm_lock_S (CC_NSF rep (s t) (e t)) = C_mfsm_lock_S (s (t+1)))"),
     REWRITE_TAC [C_mfsm_lock_S;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_mfsm_ssS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_mfsm_ssS (CC_NSF rep (s t) (e t)) = C_mfsm_ssS (s (t+1)))"),
     REWRITE_TAC [C_mfsm_ssS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_mfsm_invalidS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_mfsm_invalidS (CC_NSF rep (s t) (e t)) = C_mfsm_invalidS (s (t+1)))"),
     REWRITE_TAC [C_mfsm_invalidS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_sfsm_stateS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_sfsm_stateS (CC_NSF rep (s t) (e t)) = C_sfsm_stateS (s (t+1)))"),
     REWRITE_TAC [C_sfsm_stateS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_sfsm_DS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_sfsm_DS (CC_NSF rep (s t) (e t)) = C_sfsm_DS (s (t+1)))"),
     REWRITE_TAC [C_sfsm_DS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_sfsm_grantS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
```

```
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_sfsm_grantS (CC_NSF rep (s t) (e t)) = C_sfsm_grantS (s (t+1)))"),
     REWRITE_TAC [C_sfsm_grantS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_sfsm_rstS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_sfsm_rstS (CC_NSF rep (s t) (e t)) = C_sfsm_rstS (s (t+1)))"),
     REWRITE_TAC [C_sfsm_rstS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_sfsm_writeS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_sfsm_writeS (CC_NSF rep (s t) (e t)) = C_sfsm_writeS (s (t+1)))"),
     REWRITE_TAC [C_sfsm_writeS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_sfsm_addressedS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_sfsm_addressedS (CC_NSF rep (s t) (e t)) = C_sfsm_addressedS (s (t+1)))"),
     REWRITE_TAC [C_sfsm_addressedS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_sfsm_hlda_S_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_sfsm_hlda_S (CC_NSF rep (s t) (e t)) = C_sfsm_hlda_S (s (t+1)))"),
     REWRITE_TAC [C_sfsm_hlda_S;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_sfsm_msS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_sfsm_msS (CC_NSF rep (s t) (e t)) = C_sfsm_msS (s (t+1)))"),
     REWRITE_TAC [C_sfsm_msS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_efsm_stateS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
```

```
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_efsm_stateS (CC_NSF rep (s t) (e t)) = C_efsm_stateS (s (t+1)))"),
     REWRITE_TAC [C_efsm_stateS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_efsm_cale_S_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_efsm_cale_S (CC_NSF rep (s t) (e t)) = C_efsm_cale_S (s (t+1)))"),
     REWRITE_TAC [C_efsm_cale_S;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_efsm_last_S_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_efsm_last_S (CC_NSF rep (s t) (e t)) = C_efsm_last_S (s (t+1)))"),
     REWRITE_TAC [C_efsm_last_S;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_efsm_male_S_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_efsm_male_S (CC_NSF rep (s t) (e t)) = C_efsm_male_S (s (t+1)))"),
     REWRITE_TAC [C_efsm_male_S;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_efsm_rale_S_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_efsm_rale_S (CC_NSF rep (s t) (e t)) = C_efsm_rale_S (s (t+1)))"),
     REWRITE_TAC [C_efsm_rale_S;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_efsm_srdy_S_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_efsm_srdy_S (CC_NSF rep (s t) (e t)) = C_efsm_srdy_S (s (t+1)))"),
     REWRITE_TAC [C_efsm_srdy_S;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_efsm_rstS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
```

```
        (p :time->cc_out) .
        CBlock_GATE rep s e p
          ==>
        (C_efsm_rstS (CC_NSF rep (s t) (e t)) = C_efsm_rstS (s (t+1)))"),
     REWRITE_TAC [C_efsm_rstS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_lock_in_S_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
          ==>
        (C_lock_in_S (CC_NSF rep (s t) (e t)) = C_lock_in_S (s (t+1)))"),
     REWRITE_TAC [C_lock_in_S;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_last_in_S_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
          ==>
        (C_last_in_S (CC_NSF rep (s t) (e t)) = C_last_in_S (s (t+1)))"),
     REWRITE_TAC [C_last_in_S;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_ssS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
          ==>
        (C_ssS (CC_NSF rep (s t) (e t)) = C_ssS (s (t+1)))"),
     REWRITE_TAC [C_ssS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_clkAS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
          ==>
        (C_clkAS (CC_NSF rep (s t) (e t)) = C_clkAS (s (t+1)))"),
     REWRITE_TAC [C_clkAS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_last_out_S_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
          ==>
        (C_last_out_S (CC_NSF rep (s t) (e t)) = C_last_out_S (s (t+1)))"),
     REWRITE_TAC [C_last_out_S;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_sidle_delS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
```

```
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_sidle_delS (CC_NSF rep (s t) (e t)) = C_sidle_delS (s (t+1)))"),
     REWRITE_TAC [C_sidle_delS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_mrqt_delS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_mrqt_delS (CC_NSF rep (s t) (e t)) = C_mrqt_delS (s (t+1)))"),
     REWRITE_TAC [C_mrqt_delS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_hold_S_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_hold_S (CC_NSF rep (s t) (e t)) = C_hold_S (s (t+1)))"),
     REWRITE_TAC [C_hold_S;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_cout_0_le_delS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_cout_0_le_delS (CC_NSF rep (s t) (e t)) = C_cout_0_le_delS (s (t+1)))"),
     REWRITE_TAC [C_cout_0_le_delS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_cin_2_leS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_cin_2_leS (CC_NSF rep (s t) (e t)) = C_cin_2_leS (s (t+1)))"),
     REWRITE_TAC [C_cin_2_leS;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_mrdy_del_S_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_mrdy_del_S (CC_NSF rep (s t) (e t)) = C_mrdy_del_S (s (t+1)))"),
     REWRITE_TAC [C_mrdy_del_S;CBlock_EXP;CC_NSF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let C_iad_en_s_delS_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
```

```
         (p :time->cc_out) .
         CBlock_GATE rep s e p
            ==>
         (C_iad_en_s_delS (CC_NSF rep (s t) (e t)) = C_iad_en_s_delS (s (t+1)))"),
      REWRITE_TAC [C_iad_en_s_delS;CBlock_EXP;CC_NSF_EXP]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let C_wrdyS_THM = TAC_PROOF
    (([],
      "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
         (p :time->cc_out) .
         CBlock_GATE rep s e p
            ==>
         (C_wrdyS (CC_NSF rep (s t) (e t)) = C_wrdyS (s (t+1)))"),
      REWRITE_TAC [C_wrdyS;CBlock_EXP;CC_NSF_EXP]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let C_rrdyS_THM = TAC_PROOF
    (([],
      "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
         (p :time->cc_out) .
         CBlock_GATE rep s e p
            ==>
         (C_rrdyS (CC_NSF rep (s t) (e t)) = C_rrdyS (s (t+1)))"),
      REWRITE_TAC [C_rrdyS;CBlock_EXP;CC_NSF_EXP]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let C_parityS_THM = TAC_PROOF
    (([],
      "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
         (p :time->cc_out) .
         CBlock_GATE rep s e p
            ==>
         (C_parityS (CC_NSF rep (s t) (e t)) = C_parityS (s (t+1)))"),
      REWRITE_TAC [C_parityS;CBlock_EXP;CC_NSF_EXP]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let C_sourceS_THM = TAC_PROOF
    (([],
      "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
         (p :time->cc_out) .
         CBlock_GATE rep s e p
            ==>
         (C_sourceS (CC_NSF rep (s t) (e t)) = C_sourceS (s (t+1)))"),
      REWRITE_TAC [C_sourceS;CBlock_EXP;CC_NSF_EXP]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let C_data_inS_THM = TAC_PROOF
    (([],
      "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
         (p :time->cc_out) .
         CBlock_GATE rep s e p
            ==>
         (C_data_inS (CC_NSF rep (s t) (e t)) = C_data_inS (s (t+1)))"),
      REWRITE_TAC [C_data_inS;CBlock_EXP;CC_NSF_EXP]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let C_sizewrbeS_THM = TAC_PROOF
    (([],
      "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
```

```
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_sizewrbeS (CC_NSF rep (s t) (e t)) = C_sizewrbeS (s (t+1)))"),
    REWRITE_TAC [C_sizewrbeS;CBlock_EXP;CC_NSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let C_iad_outS_THM = TAC_PROOF
   (([],
    "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_iad_outS (CC_NSF rep (s t) (e t)) = C_iad_outS (s (t+1)))"),
    REWRITE_TAC [C_iad_outS;CBlock_EXP;CC_NSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let C_a1a0S_THM = TAC_PROOF
   (([],
    "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_a1a0S (CC_NSF rep (s t) (e t)) = C_a1a0S (s (t+1)))"),
    REWRITE_TAC [C_a1a0S;CBlock_EXP;CC_NSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let C_a3a2S_THM = TAC_PROOF
   (([],
    "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_a3a2S (CC_NSF rep (s t) (e t)) = C_a3a2S (s (t+1)))"),
    REWRITE_TAC [C_a3a2S;CBlock_EXP;CC_NSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let C_iad_inS_THM = TAC_PROOF
   (([],
    "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_iad_inS (CC_NSF rep (s t) (e t)) = C_iad_inS (s (t+1)))"),
    REWRITE_TAC [C_iad_inS;CBlock_EXP;CC_NSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let C_wrS_THM = TAC_PROOF
   (([],
    "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_wrS (CC_NSF rep (s t) (e t)) = C_wrS (s (t+1)))"),
    REWRITE_TAC [C_wrS;CBlock_EXP;CC_NSF_EXP]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let I_cgnt_O_THM = TAC_PROOF
   (([],
    "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
```

```
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (I_cgnt_O (CC_OF rep (s t) (e t)) = I_cgnt_O (p t))"),
     REWRITE_TAC [I_cgnt_O;CBlock_EXP;CC_OF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let I_mrdy_out_O_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (I_mrdy_out_O (CC_OF rep (s t) (e t)) = I_mrdy_out_O (p t))"),
     REWRITE_TAC [I_mrdy_out_O;CBlock_EXP;CC_OF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let I_hold_O_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (I_hold_O (CC_OF rep (s t) (e t)) = I_hold_O (p t))"),
     REWRITE_TAC [I_hold_O;CBlock_EXP;CC_OF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let I_rale_out_O_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (I_rale_out_O (CC_OF rep (s t) (e t)) = I_rale_out_O (p t))"),
     REWRITE_TAC [I_rale_out_O;CBlock_EXP;CC_OF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let I_male_out_O_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (I_male_out_O (CC_OF rep (s t) (e t)) = I_male_out_O (p t))"),
     REWRITE_TAC [I_male_out_O;CBlock_EXP;CC_OF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let I_last_out_O_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (I_last_out_O (CC_OF rep (s t) (e t)) = I_last_out_O (p t))"),
     REWRITE_TAC [I_last_out_O;CBlock_EXP;CC_OF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let I_srdy_out_O_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
```

```
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (I_srdy_out_O (CC_OF rep (s t) (e t)) = I_srdy_out_O (p t))"),
     REWRITE_TAC [I_srdy_out_O;CBlock_EXP;CC_OF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let I_ad_outO_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (I_ad_outO (CC_OF rep (s t) (e t)) = I_ad_outO (p t))"),
     REWRITE_TAC [I_ad_outO;CBlock_EXP;CC_OF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let I_be_out_O_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (I_be_out_O (CC_OF rep (s t) (e t)) = I_be_out_O (p t))"),
     REWRITE_TAC [I_be_out_O;CBlock_EXP;CC_OF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let CB_rqt_out_O_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (CB_rqt_out_O (CC_OF rep (s t) (e t)) = CB_rqt_out_O (p t))"),
     REWRITE_TAC [CB_rqt_out_O;CBlock_EXP;CC_OF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let CB_ms_outO_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (CB_ms_outO (CC_OF rep (s t) (e t)) = CB_ms_outO (p t))"),
     REWRITE_TAC [CB_ms_outO;CBlock_EXP;CC_OF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let CB_ss_outO_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (CB_ss_outO (CC_OF rep (s t) (e t)) = CB_ss_outO (p t))"),
     REWRITE_TAC [CB_ss_outO;CBlock_EXP;CC_OF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let CB_ad_outO_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
```

```
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (CB_ad_outO (CC_OF rep (s t) (e t)) = CB_ad_outO (p t))"),
     REWRITE_TAC [CB_ad_outO;CBlock_EXP;CC_OF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[DE_MORGAN_THM]
    );;

let C_ss_outO_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (C_ss_outO (CC_OF rep (s t) (e t)) = C_ss_outO (p t))"),
     REWRITE_TAC [C_ss_outO;CBlock_EXP;CC_OF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let Disable_writesO_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (Disable_writesO (CC_OF rep (s t) (e t)) = Disable_writesO (p t))"),
     REWRITE_TAC [Disable_writesO;CBlock_EXP;CC_OF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let CB_parityO_THM = TAC_PROOF
    (([],
     "! (t :time) (rep :^REP_ty) (s :time->cc_state) (e :time->cc_env)
        (p :time->cc_out) .
        CBlock_GATE rep s e p
           ==>
        (CB_parityO (CC_OF rep (s t) (e t)) = CB_parityO (p t))"),
     REWRITE_TAC [CB_parityO;CBlock_EXP;CC_OF_EXP]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
    );;

close_theory();;
```

3.5 SU-Cont Clock-Level Verification

The theory *sclock_ver* and file *sc_thms.ml* contain the SU-Cont clock-level correctness proof.

```
%-----------------------------------------------------------------------------

   File:          sclock_ver.ml

   Author:        (c) D.A. Fura 1992-93

   Date:          4 March 1993

-----------------------------------------------------------------------------%

set_flag ('timing', true);;

set_search_path (search_path() @ ['/home/elvis6/dfura/ftep/piu/hol/sucont/';
                                  '/home/elvis6/dfura/ftep/piu/hol/lib/';
                                  '/home/elvis6/dfura/hol/ml/';
```

```
                                     '/home/elvis6/dfura/hol/Library/tools/'
                                    ]);;

system 'rm sclock_ver.th';;

new_theory 'sclock_ver';;

loadf 'aux_defs';;

map new_parent ['wordn_def';'array_def'];;

map load_parent ['sclock_def';'sblock_def';'piuaux_def';'gates_def1';
                 'latches_def';'ffs_def';'counters_def';'saux_def'];;

new_type_abbrev ('time',":num");;

let SClockNSF_REW = theorem 'sclock_def' 'SClockNSF_REW';;
let SClockOF_REW = theorem 'sclock_def' 'SClockOF_REW';;

loadt 'sc_thms.ml';;

let S_Clock_Correct = prove_thm
   ('S_Clock_Correct',
    "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
       SBlock_GATE s e p
          ==>
       SCSet_Correct s e p",
    REPEAT STRIP_TAC
    THEN REWRITE_TAC [SCSet_Correct]
    THEN INDUCT_THEN (prove_induction_thm SCI) ASSUME_TAC
    THEN GEN_TAC
    THEN REWRITE_TAC [SC_Correct;SC_Exec;SC_PreC;SC_PostC]
    THEN CONJ_TAC
    THENL [
       % Subgoal 1: "s(t + 1) = SC_NSF(s t)(e t)" %
       SUBST_TAC [SPEC "(s (t+1)):s_state" State_Selectors_Work]
       THEN IMP_RES_TAC (SYM_RULE S_fsm_stateS_THM)
       THEN IMP_RES_TAC (SYM_RULE S_fsm_rstS_THM)
       THEN IMP_RES_TAC (SYM_RULE S_fsm_delay6S_THM)
       THEN IMP_RES_TAC (SYM_RULE S_fsm_delay17S_THM)
       THEN IMP_RES_TAC (SYM_RULE S_fsm_bothbadS_THM)
       THEN IMP_RES_TAC (SYM_RULE S_fsm_bypassS_THM)
       THEN IMP_RES_TAC (SYM_RULE S_soft_shotS_THM)
       THEN IMP_RES_TAC (SYM_RULE S_soft_shot_delS_THM)
       THEN IMP_RES_TAC (SYM_RULE S_soft_cntS_THM)
       THEN IMP_RES_TAC (SYM_RULE S_delayS_THM)
       THEN IMP_RES_TAC (SYM_RULE S_instartS_THM)
       THEN IMP_RES_TAC (SYM_RULE S_bad_cpu0S_THM)
       THEN IMP_RES_TAC (SYM_RULE S_bad_cpu1S_THM)
       THEN IMP_RES_TAC (SYM_RULE S_reset_cpu0S_THM)
       THEN IMP_RES_TAC (SYM_RULE S_reset_cpu1S_THM)
       THEN IMP_RES_TAC (SYM_RULE S_cpu_bistS_THM)
       THEN IMP_RES_TAC (SYM_RULE S_pmm_failS_THM)
       THEN IMP_RES_TAC (SYM_RULE S_cpu0_failS_THM)
       THEN IMP_RES_TAC (SYM_RULE S_cpu1_failS_THM)
       THEN IMP_RES_TAC (SYM_RULE S_piu_failS_THM)
       THEN ASM_REWRITE_TAC
              [SPEC "SC_NSF ((s:time->s_state) t) ((e:time->s_env) t)"
                    (SYM_RULE State_Selectors_Work)]
    ;
       % Subgoal 2: "p t = SC_OF(s t)(e t)" %
       SUBST_TAC [SPEC "((p:time->s_out) t)" Out_Selectors_Work]
       THEN IMP_RES_TAC (SYM_RULE S_StateO_THM)
       THEN IMP_RES_TAC (SYM_RULE Reset_cportO_THM)
       THEN IMP_RES_TAC (SYM_RULE Disable_intO_THM)
       THEN IMP_RES_TAC (SYM_RULE Reset_piuO_THM)
       THEN IMP_RES_TAC (SYM_RULE Reset_cpu0O_THM)
       THEN IMP_RES_TAC (SYM_RULE Reset_cpu1O_THM)
       THEN IMP_RES_TAC (SYM_RULE Cpu_bistO_THM)
       THEN IMP_RES_TAC (SYM_RULE Piu_failO_THM)
       THEN IMP_RES_TAC (SYM_RULE Cpu0_failO_THM)
       THEN IMP_RES_TAC (SYM_RULE Cpu1_failO_THM)
```

```
         THEN IMP_RES_TAC (SYM_RULE Pmm_fail0_THM)
         THEN ASM_REWRITE_TAC
                  [SPEC "SC_OF ((s:time->s_state) t) ((e:time->s_env) t)"
                        (SYM_RULE Out_Selectors_Work)]
      ]
    );;

close_theory();;

%------------------------------------------------------------------------

   File:          sc_thms.ml

   Author:        (c) D.A. Fura 1992-93

   Date:          4 March 1993

------------------------------------------------------------------------%

let S_fsm_stateS_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
       SBlock_GATE s e p
          ==>
       (S_fsm_stateS (SC_NSF (s t) (e t)) = S_fsm_stateS (s (t+1)))"),
    REWRITE_TAC [S_fsm_stateS;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let S_fsm_rstS_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
       SBlock_GATE s e p
          ==>
       (S_fsm_rstS (SC_NSF (s t) (e t)) = S_fsm_rstS (s (t+1)))"),
    REWRITE_TAC [S_fsm_rstS;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let S_fsm_delay6S_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
       SBlock_GATE s e p
          ==>
       (S_fsm_delay6S (SC_NSF (s t) (e t)) = S_fsm_delay6S (s (t+1)))"),
    REWRITE_TAC [S_fsm_delay6S;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let S_fsm_delay17S_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
       SBlock_GATE s e p
          ==>
       (S_fsm_delay17S (SC_NSF (s t) (e t)) = S_fsm_delay17S (s (t+1)))"),
    REWRITE_TAC [S_fsm_delay17S;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let S_fsm_bothbadS_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
       SBlock_GATE s e p
          ==>
       (S_fsm_bothbadS (SC_NSF (s t) (e t)) = S_fsm_bothbadS (s (t+1)))"),
    REWRITE_TAC [S_fsm_bothbadS;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
```

```
  );;

let S_fsm_bypassS_THM = TAC_PROOF
  (([],
   "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
      SBlock_GATE s e p
         ==>
      (S_fsm_bypassS (SC_NSF (s t) (e t)) = S_fsm_bypassS (s (t+1)))"),
   REWRITE_TAC [S_fsm_bypassS;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC[]
  );;

let S_soft_shotS_THM = TAC_PROOF
  (([],
   "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
      SBlock_GATE s e p
         ==>
      (S_soft_shotS (SC_NSF (s t) (e t)) = S_soft_shotS (s (t+1)))"),
   REWRITE_TAC [S_soft_shotS;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC[]
  );;

let S_soft_shot_delS_THM = TAC_PROOF
  (([],
   "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
      SBlock_GATE s e p
         ==>
      (S_soft_shot_delS (SC_NSF (s t) (e t)) = S_soft_shot_delS (s (t+1)))"),
   REWRITE_TAC [S_soft_shot_delS;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC[]
  );;

let S_soft_cntS_THM = TAC_PROOF
  (([],
   "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
      SBlock_GATE s e p
         ==>
      (S_soft_cntS (SC_NSF (s t) (e t)) = S_soft_cntS (s (t+1)))"),
   REWRITE_TAC [S_soft_cntS;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC[]
  );;

let S_delayS_THM = TAC_PROOF
  (([],
   "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
      SBlock_GATE s e p
         ==>
      (S_delayS (SC_NSF (s t) (e t)) = S_delayS (s (t+1)))"),
   REWRITE_TAC [S_delayS;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC[]
  );;

let S_instartS_THM = TAC_PROOF
  (([],
   "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
      SBlock_GATE s e p
         ==>
      (S_instartS (SC_NSF (s t) (e t)) = S_instartS (s (t+1)))"),
   REWRITE_TAC [S_instartS;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC[]
  );;

let S_bad_cpu0S_THM = TAC_PROOF
  (([],
   "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
      SBlock_GATE s e p
```

```
            ==>
         (S_bad_cpu0S (SC_NSF (s t) (e t)) = S_bad_cpu0S (s (t+1)))"),
      REWRITE_TAC [S_bad_cpu0S;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let S_bad_cpu1S_THM = TAC_PROOF
     (([],
      "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
         SBlock_GATE s e p
            ==>
         (S_bad_cpu1S (SC_NSF (s t) (e t)) = S_bad_cpu1S (s (t+1)))"),
      REWRITE_TAC [S_bad_cpu1S;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let S_reset_cpu0S_THM = TAC_PROOF
     (([],
      "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
         SBlock_GATE s e p
            ==>
         (S_reset_cpu0S (SC_NSF (s t) (e t)) = S_reset_cpu0S (s (t+1)))"),
      REWRITE_TAC [S_reset_cpu0S;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let S_reset_cpu1S_THM = TAC_PROOF
     (([],
      "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
         SBlock_GATE s e p
            ==>
         (S_reset_cpu1S (SC_NSF (s t) (e t)) = S_reset_cpu1S (s (t+1)))"),
      REWRITE_TAC [S_reset_cpu1S;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let S_cpu_bistS_THM = TAC_PROOF
     (([],
      "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
         SBlock_GATE s e p
            ==>
         (S_cpu_bistS (SC_NSF (s t) (e t)) = S_cpu_bistS (s (t+1)))"),
      REWRITE_TAC [S_cpu_bistS;SBlock;DFFB_GATE;DLatB_GATE;AND3_GATE;ASel;BSel;sig;
                   (EXPAND_LET_RULE SClockNSF_REW)]
      THEN REPEAT GEN_TAC
      THEN BETA_TAC
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let S_pmm_failS_THM = TAC_PROOF
     (([],
      "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
         SBlock_GATE s e p
            ==>
         (S_pmm_failS (SC_NSF (s t) (e t)) = S_pmm_failS (s (t+1)))"),
      REWRITE_TAC [S_pmm_failS;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
     );;

let S_cpu0_failS_THM = TAC_PROOF
     (([],
      "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
         SBlock_GATE s e p
            ==>
         (S_cpu0_failS (SC_NSF (s t) (e t)) = S_cpu0_failS (s (t+1)))"),
      REWRITE_TAC [S_cpu0_failS;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
```

```
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let S_cpu1_failS_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
       SBlock_GATE s e p
          ==>
       (S_cpu1_failS (SC_NSF (s t) (e t)) = S_cpu1_failS (s (t+1)))"),
    REWRITE_TAC [S_cpu1_failS;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let S_piu_failS_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
       SBlock_GATE s e p
          ==>
       (S_piu_failS (SC_NSF (s t) (e t)) = S_piu_failS (s (t+1)))"),
    REWRITE_TAC [S_piu_failS;SBlock_EXP;(EXPAND_LET_RULE SClockNSF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let S_stateO_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
       SBlock_GATE s e p
          ==>
       (S_stateO (SC_OF (s t) (e t)) = S_stateO (p t))"),
    REWRITE_TAC [S_stateO;SBlock_EXP;(EXPAND_LET_RULE SClockOF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let Reset_cportO_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
       SBlock_GATE s e p
          ==>
       (Reset_cportO (SC_OF (s t) (e t)) = Reset_cportO (p t))"),
    REWRITE_TAC [Reset_cportO;SBlock_EXP;(EXPAND_LET_RULE SClockOF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let Disable_intO_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
       SBlock_GATE s e p
          ==>
       (Disable_intO (SC_OF (s t) (e t)) = Disable_intO (p t))"),
    REWRITE_TAC [Disable_intO;SBlock_EXP;(EXPAND_LET_RULE SClockOF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let Reset_piuO_THM = TAC_PROOF
   (([],
    "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
       SBlock_GATE s e p
          ==>
       (Reset_piuO (SC_OF (s t) (e t)) = Reset_piuO (p t))"),
    REWRITE_TAC [Reset_piuO;SBlock_EXP;(EXPAND_LET_RULE SClockOF_REW)]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let Reset_cpu0O_THM = TAC_PROOF
   (([],
       "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
          SBlock_GATE s e p
             ==>
          (Reset_cpu0O (SC_OF (s t) (e t)) = Reset_cpu0O (p t))"),
       REWRITE_TAC [Reset_cpu0O;SBlock_EXP;(EXPAND_LET_RULE SClockOF_REW)]
       THEN REPEAT STRIP_TAC
       THEN ASM_REWRITE_TAC[]
      );;

let Reset_cpu1O_THM = TAC_PROOF
      (([],
       "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
          SBlock_GATE s e p
             ==>
          (Reset_cpu1O (SC_OF (s t) (e t)) = Reset_cpu1O (p t))"),
       REWRITE_TAC [Reset_cpu1O;SBlock_EXP;(EXPAND_LET_RULE SClockOF_REW)]
       THEN REPEAT STRIP_TAC
       THEN ASM_REWRITE_TAC[]
      );;

let Cpu_bistO_THM = TAC_PROOF
      (([],
       "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
          SBlock_GATE s e p
             ==>
          (Cpu_bistO (SC_OF (s t) (e t)) = Cpu_bistO (p t))"),
       REWRITE_TAC [Cpu_bistO;SBlock_EXP;(EXPAND_LET_RULE SClockOF_REW)]
       THEN REPEAT STRIP_TAC
       THEN ASM_REWRITE_TAC[]
      );;

let Piu_failO_THM = TAC_PROOF
      (([],
       "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
          SBlock_GATE s e p
             ==>
          (Piu_failO (SC_OF (s t) (e t)) = Piu_failO (p t))"),
       REWRITE_TAC [Piu_failO;SBlock_EXP;(EXPAND_LET_RULE SClockOF_REW)]
       THEN REPEAT STRIP_TAC
       THEN ASM_REWRITE_TAC[]
      );;

let Cpu0_failO_THM = TAC_PROOF
      (([],
       "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
          SBlock_GATE s e p
             ==>
          (Cpu0_failO (SC_OF (s t) (e t)) = Cpu0_failO (p t))"),
       REWRITE_TAC [Cpu0_failO;SBlock_EXP;(EXPAND_LET_RULE SClockOF_REW)]
       THEN REPEAT STRIP_TAC
       THEN ASM_REWRITE_TAC[]
      );;

let Cpu1_failO_THM = TAC_PROOF
      (([],
       "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
          SBlock_GATE s e p
             ==>
          (Cpu1_failO (SC_OF (s t) (e t)) = Cpu1_failO (p t))"),
       REWRITE_TAC [Cpu1_failO;SBlock_EXP;(EXPAND_LET_RULE SClockOF_REW)]
       THEN REPEAT STRIP_TAC
       THEN ASM_REWRITE_TAC[]
      );;

let Pmm_failO_THM = TAC_PROOF
      (([],
       "! (t :time) (s :time->s_state) (e :time->s_env) (p :time->s_out) .
          SBlock_GATE s e p
             ==>
          (Pmm_failO (SC_OF (s t) (e t)) = Pmm_failO (p t))"),
       REWRITE_TAC [Pmm_failO;SBlock_EXP;(EXPAND_LET_RULE SClockOF_REW)]
       THEN REPEAT STRIP_TAC
```

```
 THEN ASM_REWRITE_TAC[]
);;
```

4 Requirements Verification

This file contains the HOL listings for the major portion of the P-Port transaction-level verification. The theory *ptrans_ver* contains the top-level correctness statement for the P-Port. The files *pt_thms1.ml* and *pt_thms2.ml* contain the bulk of the theorems used in the correctess proof.

```
%----------------------------------------------------------------------------

   File:      ptrans_ver.ml

   Author:    (c) D.A. Fura 1992-93

   Date:      7 March 1993

----------------------------------------------------------------------------%

set_search_path (search_path() @ ['/home/elvis6/dfura/ftep/piu/hol/pport/';
                                  '/home/elvis6/dfura/ftep/piu/hol/lib/';
                                  '/home/elvis6/dfura/hol/Library/tools/';
                                  '/home/elvis6/dfura/hol/ml/'
                                 ]);;

set_flag ('timing',true);;

system 'rm ptrans_ver.th';;

new_theory 'ptrans_ver';;

load_library 'reduce';;

loadf 'aux_defs';;

map load_parent ['piuaux_def';'ptauxp_def';'paux_def';'array_def';'wordn_def';
                 'busn_def';'templogic_def';'ptransp_def';'pclock_def';
                 'ptabs_def';'ineq';'assoc';'cond'];;

new_type_abbrev ('time',":num");;
new_type_abbrev ('timeT',":num");;
new_type_abbrev ('timeC',":num");;

let PT_WriteOF_EXP = EXPAND_LET_RULE (definition 'ptrans_def' 'PT_WriteOF');;
let PTAbs_EXP =
   EXPAND_LET_RULE (definition 'ptabs_def' 'PTAbs');;
let PB_Slave_EXP = EXPAND_LET_RULE (definition 'ptabs_def' 'PB_Slave');;
let IB_PMaster_EXP = EXPAND_LET_RULE (definition 'ptabs_def' 'IB_PMaster');;
let PStateAbs_EXP = EXPAND_LET_RULE (definition 'ptabs_def' 'PStateAbs');;

set_flag ('print_all_subgoals',false);;

loadf 'pt_tacs.ml';;
loadf 'pt_defs.ml';;
loadt 'pt_thms1.ml';;
loadt 'pt_thms2.ml';;

----------------------------------------------
   (PT_Write, IB_Addr_outO) Theorem:
----------------------------------------------
let ADDR_WRITE = TAC_PROOF
   (([],
    "! (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out) .
     PCSet_Correct s' e' p' ==>
       PTAbsSet s e p s' e' p' ==>
         PT_Exec PT_Write s e p t ==>
           PT_PreC PT_Write s e p t ==>
             (IB_Addr_outO (PT_WriteOF (s t) (e t)) = IB_Addr_outO (p t))"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC ABS_SET_IMP_ABS
    THEN NRULE_ASSUM_TAC
           ("!pti t. PTAbs pti s e p t s' e' p'",
            ((SPECL ["PT_Write";"t:timeT"]) o (REWRITE_RULE [PTAbs])))
    THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
    THEN RES_TAC
    THEN RES_TAC
    THEN POP_ASSUM (\thm. ALL_TAC) %KEEP%
    THEN POP_ASSUM (\thm. ALL_TAC) %KEEP%
    THEN IMP_RES_TAC NTH_IBUS_TRANS_EXISTS
    THEN IMP_RES_TAC NTH_TIME_TRUE_X_IMP_X
    THEN POP_ASSUM (\thm. ALL_TAC) %KEEP%
    THEN RES_TAC
    THEN POP_ASSUM (\thm. ALL_TAC) %KEEP%
    THEN POP_ASSUM (\thm. ALL_TAC) %KEEP%
    THEN IMP_RES_TAC PB_Addr_in_ISO
    THEN IMP_RES_TAC IB_Addr_out_ISO
    THEN IMP_RES_TAC I_ad_out_ISO
    THEN IMP_RES_TAC (REWRITE_RULE [New_State_Is_PA] IBUS_ALE_IMP_PA)
    THEN ASM_REWRITE_TAC [prove_constructors_distinct pfsm_ty_Axiom;
                          PT_WriteOF_EXP;IB_Addr_outO; wordnVAL_BUSN_IDENT;
                          EXTRACT_ADDR]
    THEN POP_ASSUM (\thm. ALL_TAC)
    THEN POP_ASSUM (\thm. ALL_TAC)
    THEN ASM_CASES_TAC "P_rqtS (s' (ti':timeC))"
    THEN ASM_REWRITE_TAC[]
    THENL [
       % Subgoal 1: [ "P_rqtS(s' ti')" ] %
       IMP_RES_TAC
         (REWRITE_RULE [DE_MORGAN_THM] P_RQT_TRUE_ON_TI'_IMP_DELAY_CONDS)
       THEN IMP_RES_TAC (REWRITE_RULE [DE_MORGAN_THM] ALE_SIG_IB_TRUE_AFTER_TP')
       THEN IMP_RES_TAC NEXT_IBUS_TRANS_IS_NTH
       THEN IMP_RES_TAC TI'_AFTER_TP'
       THEN
         (SUBGOAL_THEN "(ti':timeC) = ti''" ASSUME_TAC
          THENL [
             IMP_RES_TAC TRUE_EVENT_TIMES_EQUAL
          ;
             IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LT_IMP_LE_PRE)
             THEN IMP_RES_TAC STABLE_FALSE_THEN
             THEN ASSUME_TAC (SPECL ["ti'':timeC";"1"] SUB_LESS_EQ)
             THEN IMP_RES_TAC NEW_P_ADDR_STABLE_FROM_TP'_TO_TI'
             THEN IMP_RES_TAC M_LESS_0_LESS
             THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LT_IMP_SUC_LE)
             THEN POP_ASSUM_LIST (MAP_EVERY (\thm. ASSUME_TAC (REDUCE_RULE thm)))
             THEN IMP_RES_TAC (SPECL ["ti'':timeC";"1"] SUB_ADD)
             THEN ASM_REWRITE_ASSUM_TAC
                     ("P_addrS(s'((ti'' - 1) + 1)) =
                       SUBARRAY(FST(L_ad_inE(e' (tp':timeC))))(25,0)",[])
             THEN ASSUME_TAC (SPEC "25" LESS_EQ_REFL)
             THEN IMP_RES_TAC (ISPEC "FST(L_ad_inE(e' (tp':timeC)))"
                                SUB_SUBARRAY)
             THEN ASM_REWRITE_TAC[]
          ])
    ;
       % Subgoal 2: [ "~P_rqtS(s' ti')" ] %
       IMP_RES_TAC P_RQT_FALSE_ON_TI'_IMP_FLOWTHRU_CONDS
       THEN IMP_RES_TAC TRANS_TIMES_EQUAL
       THEN ASSUME_TAC (SPEC "25" LESS_EQ_REFL)
       THEN IMP_RES_TAC (ISPEC "FST(L_ad_inE(e' (ti':timeC)))" SUB_SUBARRAY)
       THEN ASM_REWRITE_TAC[]
    ]
   );;
%
-----------------------------------------------
   Main Theorem:
-----------------------------------------------
g "! (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
     (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out) .
   PCSet_Correct s' e' p' ==>
      PTAbsSet s e p s' e' p' ==>
         PTSet_Correct s e p";;
    REPEAT STRIP_TAC
    THEN REWRITE_TAC [PTSet_Correct;PT_Correct;PT_PostC]
    THEN INDUCT_THEN (prove_induction_thm PTI) ASSUME_TAC
    THEN REWRITE_TAC [SYM_RULE (prove_constructors_distinct PTI)]
    THEN REPEAT STRIP_TAC

4 subgoals:

"(s(t + 1) = PT_WriteNSF_A(s t)(e t)) \/
 (s(t + 1) = PT_WriteNSF_H(s t)(e t))"
    [ "PCSet_Correct s' e' p'" ]
    [ "PTAbsSet s e p s' e' p'" ]
    [ "PT_Exec PT_Write s e p t" ]
    [ "PT_PreC PT_Write s e p t" ]

"p t = PT_WriteOF(s t)(e t)"
    [ "PCSet_Correct s' e' p'" ]
    [ "PTAbsSet s e p s' e' p'" ]
    [ "PT_Exec PT_Write s e p t" ]
    [ "PT_PreC PT_Write s e p t" ]

"(s(t + 1) = PT_ReadNSF_A(s t)(e t)) \/
 (s(t + 1) = PT_ReadNSF_H(s t)(e t))"
    [ "PCSet_Correct s' e' p'" ]
    [ "PTAbsSet s e p s' e' p'" ]
    [ "PT_Exec PT_Read s e p t" ]
    [ "PT_PreC PT_Read s e p t" ]

"p t = PT_ReadOF(s t)(e t)"
    [ "PCSet_Correct s' e' p'" ]
    [ "PTAbsSet s e p s' e' p'" ]
    [ "PT_Exec PT_Read s e p t" ]
    [ "PT_PreC PT_Read s e p t" ]


%------------------------------------------------------------------------------

   File:     pt_thms1.ml

   Author:   (c) D.A. Fura 1992-93

   Date:     7 March 1993

   Theorems used in the P-Port trans-level proof.

------------------------------------------------------------------------------%

% [PJW] %
let FIRST_EXISTS = TAC_PROOF
   (([],
    "! (x :time->bool) (t0 t9 :time) .
     (? t. x t /\ t0 <= t /\ t <= t9) ==>
        (? u. t0 <= u /\ u <= t9 /\ STABLE_FALSE_THEN_TRUE x (t0,u))"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC (BETA_RULE (
                  SPEC "\t. x t /\ t0 <= t /\ t <= t9" WOP))
    THEN EXISTS_TAC "n':num"
    THEN ASM_REWRITE_TAC [STABLE_FALSE_THEN_TRUE]
    THEN REPEAT STRIP_TAC
    THEN RES_TAC
    THEN IMP_RES_TAC (
         IMP_TRANS
            (SPECL ["t':num";"n':num";"t9:num"] LESS_LESS_EQ_TRANS)
            (SPECL ["t':num";"t9:num"] LESS_IMP_LESS_OR_EQ))
    THEN RES_TAC
   );;

let FIRST_EXISTS1 = mk_thm
   ([],
    "! (x :time->bool) (t t0 t9 :time) .
     x t /\ t0 <= t /\ t <= t9 ==>
        (? u. t0 <= u /\ u <= t9 /\ STABLE_FALSE_THEN_TRUE x (t0,u))"
   );;
```

```
let DISJOINT_OR = TAC_PROOF
   (([],
     "! (a b :bool) . a \/ b = a \/ ~a /\ b"),
    REPEAT GEN_TAC
    THEN BOOL_CASES_TAC "a:bool"
    THEN BOOL_CASES_TAC "b:bool"
    THEN ASM_REWRITE_TAC[]
   );;

let RM_NORESET = TAC_PROOF
   (([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) .
     PT_Exec pti s e p t ==> (Rst_Opcode_inE (e t) = RM_NoReset)"),
    REWRITE_TAC [PT_Exec]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let PBM_REQUEST = TAC_PROOF
   (([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) .
     PT_Exec pti s e p t ==>
       ((PB_Opcode_inE (e t) = PBM_WriteLM) \/
        (PB_Opcode_inE (e t) = PBM_WritePIU) \/
        (PB_Opcode_inE (e t) = PBM_WriteCB) \/
        (PB_Opcode_inE (e t) = PBM_ReadLM) \/
        (PB_Opcode_inE (e t) = PBM_ReadPIU) \/
        (PB_Opcode_inE (e t) = PBM_ReadCB))"),
    REWRITE_TAC [PT_Exec]
    THEN REPEAT GEN_TAC
    THEN ASM_CASES_TAC "pti = PT_Write"
    THEN ASM_REWRITE_TAC[]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let IBS_READY = TAC_PROOF
   (([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) .
     PT_Exec pti s e p t ==> (IB_Opcode_inE (e t) = IBS_Ready)"),
    REWRITE_TAC [PT_Exec]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let IBAS_READY = TAC_PROOF
   (([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) .
     PT_Exec pti s e p t ==> (IBA_Opcode_inE (e t) = IBAS_Ready)"),
    REWRITE_TAC [PT_Exec]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let ABS_SET_IMP_ABS = TAC_PROOF
   (([],
     "PTAbsSet s e p s' e' p' ==>
        (!(t:timeT)(pti:PTI). PTAbs pti s e p t s' e' p')"),
    REWRITE_TAC [PTAbsSet]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let RST_FALSE = TAC_PROOF
   (([],
     "! (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out) (t :timeT)
        (e' :timeC->pc_env) (pti :PTI) .
```

```
          PT_Exec pti s e p t ==>
             Rst_Slave pti e t e' ==>
                (!u':timeC. (SND(RstE (e' u')) = F))"),
        REWRITE_TAC [PT_Exec;Rst_Slave;BSel]
        THEN REPEAT GEN_TAC
        THEN STRIP_TAC
        THEN ASM_REWRITE_TAC []
        THEN ASM_CASES_TAC "!u':timeC. ~SND(RstE(e' u'))"
        THEN ASM_REWRITE_TAC[prove_constructors_distinct rmop]
       );;

let ALE_SIG_PB_INIT_FALSE = TAC_PROOF
   (([],
    "! (tp' :timeC) (e' :timeC->pc_env) .
     NTH_TIME_TRUE 0 (ale_sig_pb e') 0 tp' ==>
        (!t':timeC.
          t' < tp' ==> ~(~SND(L_ads_E (e' t')) /\ SND(L_den_E (e' t'))))"),
    REWRITE_TAC [NTH_TIME_TRUE;STABLE_FALSE_THEN_TRUE;BSel;ale_sig_pb]
    THEN BETA_TAC
    THEN REPEAT STRIP_TAC
    THEN ASSUME_TAC (SPEC "t':timeC" ZERO_LESS_EQ)
    THEN RES_TAC
   );;

let NTH_ALE_SIG_PB_TRUE = TAC_PROOF
   (([],
    "! (n :num) (tp' :timeC) (e' :timeC->pc_env) .
     NTH_TIME_TRUE n (ale_sig_pb e') 0 tp' ==>
        (~SND(L_ads_E (e' tp')) /\ SND(L_den_E (e' tp')))"),
    INDUCT_TAC
    THEN REWRITE_TAC [NTH_TIME_TRUE;STABLE_FALSE_THEN_TRUE;ale_sig_pb;BSel]
    THEN BETA_TAC
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
    THEN RES_TAC
   );;

let ALE_SIG_IB_TRUE = TAC_PROOF
   (([],
    "! (n :num) (ti' :timeC) (p' :timeC->pc_out) .
     ale_sig_ib p' ti' ==>
        (SND(I_hlda_O (p' ti')) /\ ((SND(I_male_O (p' ti')) = LO) \/
                                    (SND(I_rale_O (p' ti')) = LO) \/
                                    (SND(I_cale_O (p' ti')) = F)))"),
    INDUCT_TAC
    THEN REWRITE_TAC [ale_sig_ib;BSel]
    THEN BETA_TAC
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
    THEN RES_TAC
   );;

let NTH_ALE_SIG_IB_TRUE = TAC_PROOF
   (([],
    "! (n :num) (ti' :timeC) (p' :timeC->pc_out) .
     NTH_TIME_TRUE n (ale_sig_ib p') 0 ti' ==>
        (SND(I_hlda_O (p' ti')) /\ ((SND(I_male_O (p' ti')) = LO) \/
                                    (SND(I_rale_O (p' ti')) = LO) \/
                                    (SND(I_cale_O (p' ti')) = F)))"),
    INDUCT_TAC
    THEN REWRITE_TAC [NTH_TIME_TRUE;STABLE_FALSE_THEN_TRUE;ale_sig_ib;BSel]
    THEN BETA_TAC
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
    THEN RES_TAC
   );;

let PB_REQUEST_ASSUMPS = TAC_PROOF
   (([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (e' :timeC->pc_env) (p' :timeC->pc_out) (tp' :timeC) .
     PT_Exec pti s e p t ==>
```

```
        PB_Slave pti e p t e' p' tp' ==>
          (!u'. LESS_THAN_N_TIMES_FALSE
                  (VAL 1(SUBARRAY(BSel(L_ad_inE(e' tp')))(1,0)))
                  (bsig L_ready_O p') tp' u' ==>
                STABLE_FALSE(ale_sig_pb e')(tp' + 1,u'+1))"),
    REPEAT STRIP_TAC
    THEN REWRITE_ASSUM_TAC
           ("PB_Slave pti e p t e' p' tp'",[EXPAND_LET_RULE PB_Slave])
    THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
    THEN ASM_CASES_TAC
           "(!u'. LESS_THAN_N_TIMES_FALSE
                   (VAL 1(SUBARRAY(BSel(L_ad_inE(e' tp')))(1,0)))
                   (bsig L_ready_O p') tp' u' ==>
                     STABLE_FALSE(ale_sig_pb e')(tp' + 1,u'+1))"
    THEN RES_TAC
    THEN ASM_REWRITE_TAC[]
    THEN ASSUME_TAC (prove_constructors_distinct pbmop)
    THEN ASSUME_TAC (SYM_RULE (prove_constructors_distinct pbmop))
    THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
    THEN IMP_RES_TAC PBM_REQUEST
    THENL [
       UNDISCH_TAC "PB_Opcode_inE(e (t:timeT)) = PBM_WriteLM"
    ;
       UNDISCH_TAC "PB_Opcode_inE(e (t:timeT)) = PBM_WritePIU"
    ;
       UNDISCH_TAC "PB_Opcode_inE(e (t:timeT)) = PBM_WriteCB"
    ;
       UNDISCH_TAC "PB_Opcode_inE(e (t:timeT)) = PBM_ReadLM"
    ;
       UNDISCH_TAC "PB_Opcode_inE(e (t:timeT)) = PBM_ReadPIU"
    ;
       UNDISCH_TAC "PB_Opcode_inE(e (t:timeT)) = PBM_ReadCB"
    ]
    THEN ASM_REWRITE_TAC[]
   );;

let IB_READY_ASSUMPS = TAC_PROOF
   (([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (e' :timeC->pc_env) (p' :timeC->pc_out) (ti' :timeC) .
     PT_Exec pti s e p t ==>
        IB_PMaster pti e p t e' p' ti' ==>
           (?u'. STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(ti'+1,u')) /\
           (!u'. rdy_sig_ib e' p' u' ==>
                 (?v'. STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (u'+1,v')))"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC IBS_READY
    THEN IMP_RES_TAC IB_Opcode_in_ISO
    THEN ASM_REWRITE_ASSUM_TAC
           ("IB_Opcode_inE((e:timeT->pt_env) t) = IBS_Ready", [])
    THEN ASSUME_TAC (prove_constructors_distinct ibsop)
    THEN IMP_RES_TAC COND_FIRST_CHOICE
    THEN ASM_REWRITE_TAC[]
    THENL [
       EXISTS_TAC "u'':timeC"
    ;
       EXISTS_TAC "v':timeC"
    ]
    THEN ASM_REWRITE_TAC[]
   );;

let IBA_READY_ASSUMPS = TAC_PROOF
   (([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (e' :timeC->pc_env) (p' :timeC->pc_out) (t' :timeC) .
     PT_Exec pti s e p t ==>
        IBA_PMaster pti e p t e' p' ==>
          ((!u'. ?v'. STABLE_FALSE_THEN_TRUE (bsig I_hold_E e') (u',v')) /\
           (!u'. CHANGES_FALSE (bsig I_crqt_O p') u' ==>
                   (?v'. (u' < v') /\
                         STABLE_TRUE_THEN_FALSE (bsig I_cgnt_E e') (u',v'))) /\
           (!u'. BSel(I_crqt_O (p' u')) ==> BSel(I_cgnt_E (e' u'))) /\
           (!u'. ~BSel(I_cgnt_E (e' u')) ==>
                     (BSel(I_hold_E (e' u')) /\ BSel(I_hold_E (e' (u'-1))))))"),
     REPEAT STRIP_TAC
     THEN IMP_RES_TAC IBAS_READY
     THEN IMP_RES_TAC IBA_Opcode_in_ISO
     THEN ASM_REWRITE_ASSUM_TAC
           ("IBA_Opcode_inE((e:timeT->pt_env) t) = IBAS_Ready", [])
     THEN ASSUME_TAC (prove_constructors_distinct ibasop)
     THEN IMP_RES_TAC COND_FIRST_CHOICE
     THEN ASM_REWRITE_TAC[]
     THEN SPEC_ASSUM_TAC
          ("!u'. ?v'. STABLE_FALSE_THEN_TRUE(bsig I_hold_E e')(u',v')","u':timeC")
     THEN CHOOSE_ASSUM_TAC "?v'. STABLE_FALSE_THEN_TRUE(bsig I_hold_E e')(u',v')"
     THEN EXISTS_TAC "v':timeC"
     THEN ASM_REWRITE_TAC[]
    );;

let NOT_PA = TAC_PROOF
    (([], "! (x :pfsm_ty) . ~(x = PA) ==> ((x = PD) \/ (x = PH))"),
     INDUCT_THEN (prove_induction_thm pfsm_ty_Axiom) ASSUME_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let NOT_PD = TAC_PROOF
    (([], "! (x :pfsm_ty) . ~(x = PD) ==> ((x = PA) \/ (x = PH))"),
     INDUCT_THEN (prove_induction_thm pfsm_ty_Axiom) ASSUME_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let NOT_PH = TAC_PROOF
    (([], "! (x :pfsm_ty) . ~(x = PH) ==> ((x = PA) \/ (x = PD))"),
     INDUCT_THEN (prove_induction_thm pfsm_ty_Axiom) ASSUME_TAC
     THEN ASM_REWRITE_TAC[]
    );;

let PA_IMP_NOT_PD = TAC_PROOF
    (([],
      "! (s :pfsm_ty) . (s = PA) ==> ~(s = PD)"),
     GEN_TAC
     THEN DISCH_TAC
     THEN ASM_REWRITE_TAC [prove_constructors_distinct pfsm_ty_Axiom]
    );;

let PH_IMP_NOT_PD = TAC_PROOF
    (([],
      "! (s :pfsm_ty) . (s = PH) ==> ~(s = PD)"),
     GEN_TAC
     THEN DISCH_TAC
     THEN ASM_REWRITE_TAC [prove_constructors_distinct pfsm_ty_Axiom]
    );;

let PH_IMP_NOT_PA = TAC_PROOF
    (([],
      "! (s :pfsm_ty) . (s = PH) ==> ~(s = PA)"),
     GEN_TAC
     THEN DISCH_TAC
     THEN ASM_REWRITE_TAC [prove_constructors_distinct pfsm_ty_Axiom]
    );;

let NEXT_STATE_NOT_PA = TAC_PROOF
    (([],
      "! (s' :timeC->pc_state) (e' :timeC->pc_env) (t' :timeC) .
       ~New_State_Is_PA s' e' t' ==>
           (New_State_Is_PD s' e' t' \/ New_State_Is_PH s' e' t')"),
     REWRITE_TAC [New_State_Is_PA;New_State_Is_PD;New_State_Is_PH;NOT_PA]
    );;

let NEXT_STATE_NOT_PD = TAC_PROOF
    (([],
      "! (s' :timeC->pc_state) (e' :timeC->pc_env) (t' :timeC) .
       ~New_State_Is_PD s' e' t' ==>
           (New_State_Is_PA s' e' t' \/ New_State_Is_PH s' e' t')"),
    REWRITE_TAC [New_State_Is_PA;New_State_Is_PD;New_State_Is_PH;NOT_PD]
   );;

let P_RQT_INIT = TAC_PROOF
   (([],
     "! (pti :PTI) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
        (t' :timeC) .
      PCSet_Correct s' e' p' ==>
        BSel(RstE (e' t')) ==>
          ~ale_sig_pb e' t' ==>
            ~P_rqtS (s' (t'+1))"),
    REWRITE_TAC [BSel;ale_sig_pb]
    THEN BETA_TAC
    THEN REPEAT STRIP_TAC
    THEN UNDISCH_TAC "P_rqtS(s'(t' + 1))"
    THEN IMP_RES_TAC P_rqt_ISO
    THEN ASM_REWRITE_TAC[]
   );;

let P_FSM_RST_INIT = TAC_PROOF
   (([],
     "! (pti :PTI) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
        (t' :timeC) .
      PCSet_Correct s' e' p' ==>
        BSel(RstE (e' t')) ==>
          P_fsm_rstS (s' (t'+1))"),
    REWRITE_TAC [BSel]
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC P_fsm_rst_ISO
    THEN ASM_REWRITE_TAC[]
   );;

let P_FSM_STATE_INIT = TAC_PROOF
   (([],
     "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env)
        (p :timeT->pt_out) (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env)
        (p' :timeC->pc_out) (t' :timeC) .
      PCSet_Correct s' e' p' ==>
        BSel(RstE (e' t')) ==>
          ~(P_fsm_stateS (s' (t'+2)) = PD)"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC P_FSM_RST_INIT
    THEN UNDISCH_TAC "P_fsm_stateS(s'(t' + 2)) = PD"
    THEN IMP_RES_TAC
           (REWRITE_RULE
              [ASSOC_ADD_ADD1;REDUCE_CONV "1+1"]
              (SPECL ["s':timeC->pc_state";"e':timeC->pc_env";"p':timeC->pc_out";
                      "t'+1"] (GEN_ALL P_fsm_state_ISO)))
    THEN ASM_REWRITE_TAC[prove_constructors_distinct pfsm_ty_Axiom]
   );;

let P_FSM_RST_FALSE = TAC_PROOF
   (([],
     "! (t' :timeC) (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env)
        (p :timeT->pt_out) (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env)
        (p' :timeC->pc_out) (tp' :timeC) .
      PCSet_Correct s' e' p' ==>
         PT_Exec pti s e p t ==>
            Rst_Slave pti e t e' ==>
               (1 <= t') ==>
                  ~P_fsm_rstS (s' t')"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC RST_FALSE
    THEN UNDISCH_TAC "P_fsm_rstS(s' (t':timeC))"
    THEN IMP_RES_TAC (SYM_RULE (SPECL ["t':timeC";"1"] SUB_ADD))
    THEN PURE_ONCE_ASM_REWRITE_TAC[]
    THEN POP_ASSUM (\thm. ALL_TAC)
    THEN IMP_RES_TAC
           (SPECL ["s':timeC->pc_state";"e':timeC->pc_env";"p':timeC->pc_out";
                   "t'-1"] (GEN_ALL P_fsm_rst_ISO))
    THEN ASM_REWRITE_TAC[]
   );;
```

```
let I_CALE_IMP_I_CGNT = TAC_PROOF
   (([],
     "! (t' :timeC) (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env)
        (p :timeT->pt_out) (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env)
        (p' :timeC->pc_out) (ti' :timeC) .
      PCSet_Correct s' e' p' ==>
        ~SND(I_cale_O (p' t')) ==>
          ~SND(I_cgnt_E (e' t'))"),
    REPEAT GEN_TAC
    THEN DISCH_TAC
    THEN IMP_RES_TAC I_cale_ISO
    THEN ASM_REWRITE_TAC[]
    THEN DISCH_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let IBUS_ALE_IMP_PA = TAC_PROOF
   (([],
     "PCSet_Correct s' e' p' ==>
        ale_sig_ib p' ti' ==>
          New_State_Is_PA s' e' ti'"),
    REWRITE_TAC [New_State_Is_PA;ale_sig_ib;BSel]
    THEN BETA_TAC
    THEN DISCH_TAC
    THEN IMP_RES_TAC I_male_ISO
    THEN IMP_RES_TAC I_rale_ISO
    THEN IMP_RES_TAC I_cale_ISO
    THEN ASM_CASES_MATCH_RHS_TAC "PA"
    THEN IMP_RES_TAC NOT_PA
    THEN ASM_REWRITE_TAC
           [WIRE;SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom);
            prove_constructors_distinct wire;
            SYM_RULE (prove_constructors_distinct wire)]
   );;

let IBUS_ALE_IMP_NEW_P_RQT = TAC_PROOF
   (([],
     "PCSet_Correct s' e' p' ==>
        PT_Exec pti s e p t ==>
          IBA_PMaster pti e p t e' p' ==>
            ale_sig_ib p' ti' ==>
              New_P_Rqt_Is_TRUE s' e' ti'"),
    REWRITE_TAC [ale_sig_ib;BSel]
    THEN BETA_TAC
    THEN ASM_CASES_TAC "New_P_Rqt_Is_TRUE s' e' ti'"
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
    THENL [
       % Subgoal 1: [ "SND(I_male_O(p' ti')) = LO" ] %
       UNDISCH_TAC "SND(I_male_O(p' (ti':timeC))) = LO"
       THEN IMP_RES_TAC I_male_ISO
       THEN REWRITE_ASSUM_TAC
             ("~New_P_Rqt_Is_TRUE s' e' ti'",[New_P_Rqt_Is_TRUE;New_State_Is_PD])
       THEN ASM_REWRITE_TAC
              [New_P_Rqt_Is_TRUE;New_State_Is_PD;
               WIRE;SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom);
               prove_constructors_distinct wire;
               SYM_RULE (prove_constructors_distinct wire)]
       THEN COND_CASES_TAC
       THEN ASM_REWRITE_TAC
              [prove_constructors_distinct wire;
               SYM_RULE (prove_constructors_distinct wire)]
    ;
       % Subgoal 2: [ "SND(I_rale_O(p' ti')) = LO" ] %
       UNDISCH_TAC "SND(I_rale_O(p' (ti':timeC))) = LO"
       THEN IMP_RES_TAC I_rale_ISO
       THEN REWRITE_ASSUM_TAC
             ("~New_P_Rqt_Is_TRUE s' e' ti'",[New_P_Rqt_Is_TRUE;New_State_Is_PD])
       THEN ASM_REWRITE_TAC
              [New_P_Rqt_Is_TRUE;New_State_Is_PD;
               WIRE;SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom);
             prove_constructors_distinct wire;
             SYM_RULE (prove_constructors_distinct wire)]
      THEN COND_CASES_TAC
      THEN ASM_REWRITE_TAC
            [prove_constructors_distinct wire;
             SYM_RULE (prove_constructors_distinct wire)]
    ;
      % Subgoal 3: [ "~SND(I_cale_O(p' ti'))" ] %
      IMP_RES_TAC (REWRITE_RULE [BSel] IBA_READY_ASSUMPS)
      THEN SPEC_ASSUM_TAC
            ("!u':timeC. SND(I_crqt_O(p' u')) ==> SND(I_cgnt_E(e'¯u'))",
             "ti':timeC")
      THEN UNDISCH_TAC "~SND(I_cale_O(p' (ti':timeC)))"
      THEN IMP_RES_TAC I_crqt_ISO
      THEN SPEC_UNDISCH_MATCH_LHS_TAC ("I_crqt_O(p' (t:timeC))","ti':timeC")
      THEN REWRITE_ASSUM_TAC
           ("~New_P_Rqt_Is_TRUE s' e' ti'",[New_P_Rqt_Is_TRUE;New_State_Is_PD])
      THEN ASM_REWRITE_TAC[]
      THEN DISCH_TAC
      THEN RES_TAC
      THEN UNDISCH_TAC "SND(I_crqt_O(p' ti')) ==> SND(I_cgnt_E(e' (ti':timeC)))"
      THEN ASM_REWRITE_TAC[]
      THEN DISCH_TAC
      THEN IMP_RES_TAC I_cale_ISO
      THEN ASM_REWRITE_TAC[]
    ]
   );;

let IBUS_ALE_IMP_FSM_RQT = TAC_PROOF
   (([],
     "! (t' :timeC) (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env)
        (p :timeT->pt_out) (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env)
        (p' :timeC->pc_out) (ti' :timeC) .
      PCSet_Correct s' e' p' ==>
        ale_sig_ib p' ti' ==>
          PT_Exec pti s e p t ==>
            IBA_PMaster pti e p t e' p' ==>
              (P_fsm_mrqtS (s' (ti' + 1)) \/
               ~P_fsm_crqt_S (s' (ti' + 1)) /\ ~P_fsm_cgnt_S (s' (ti' + 1)))"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC ALE_SIG_IB_TRUE
    THENL [
      IMP_RES_TAC IBUS_ALE_IMP_PA
      THEN REWRITE_ASSUM_TAC ("New_State_Is_PA s' e' ti'",[New_State_Is_PA])
      THEN UNDISCH_TAC "SND(I_male_O((p':timeC->pc_out) ti')) = LO"
      THEN IMP_RES_TAC P_fsm_mrqt_ISO
      THEN IMP_RES_TAC I_male_ISO
      THEN ASM_REWRITE_TAC
              [WIRE;COND_FALSE_TRUE;COND_FALSE_CHOICES;
               prove_constructors_distinct pfsm_ty_Axiom;
               SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
      THEN ASSUME_TAC (prove_constructors_distinct wire)
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN DISCH_TAC
      THEN IMP_RES_TAC COND_SECOND_CHOICE
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. ASSUME_TAC (REWRITE_RULE [] thm)))
      THEN ASM_REWRITE_TAC[]
    ;
      IMP_RES_TAC IBUS_ALE_IMP_PA
      THEN REWRITE_ASSUM_TAC ("New_State_Is_PA s' e' ti'",[New_State_Is_PA])
      THEN UNDISCH_TAC "SND(I_rale_O((p':timeC->pc_out) ti')) = LO"
      THEN IMP_RES_TAC P_fsm_mrqt_ISO
      THEN IMP_RES_TAC I_rale_ISO
      THEN ASM_REWRITE_TAC
              [WIRE;COND_FALSE_TRUE;COND_FALSE_CHOICES;
               prove_constructors_distinct pfsm_ty_Axiom;
               SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
      THEN ASSUME_TAC (prove_constructors_distinct wire)
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN DISCH_TAC
      THEN IMP_RES_TAC COND_SECOND_CHOICE
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. ASSUME_TAC (REWRITE_RULE [] thm)))
```

```
            THEN ASM_REWRITE_TAC[]
         ;
            IMP_RES_TAC (REWRITE_RULE [BSel] IBA_READY_ASSUMPS)
            THEN NRULE_ASSUM_TAC
                    ("!u'. SND(I_crqt_O(p' (u':timeC))) ==>
                           SND(I_cgnt_E(e' (u':timeC)))",
                     (CONTRAPOS o (SPEC "ti':timeC")))
            THEN POP_ASSUM_LIST (MAP_EVERY (\thm. ASSUME_TAC (REWRITE_RULE [] thm)))
            THEN IMP_RES_TAC I_CALE_IMP_I_CGNT
            THEN RES_TAC
            THEN IMP_RES_TAC IBUS_ALE_IMP_PA
            THEN REWRITE_ASSUM_TAC ("New_State_Is_PA s' e' ti'",[New_State_Is_PA])
            THEN IMP_RES_TAC P_fsm_crqt_ISO
            THEN IMP_RES_TAC P_fsm_cgnt_ISO
            THEN IMP_RES_TAC I_cale_ISO
            THEN IMP_RES_TAC I_crqt_ISO
            THEN UNDISCH_TAC "~SND(I_cale_O((p':timeC->pc_out) ti'))"
            THEN UNDISCH_TAC "~SND(I_crqt_O((p':timeC->pc_out) ti'))"
            THEN ASM_REWRITE_TAC
                    [WIRE;COND_FALSE_TRUE;COND_FALSE_CHOICES;
                     prove_constructors_distinct pfsm_ty_Axiom;
                     SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
            THEN DISCH_TAC
            THEN ASM_REWRITE_TAC []
         ]
        );;

let P_RQT_UPTO_FIRST = TAC_PROOF
    (([],
     "! (t' :timeC) (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env)
        (p :timeT->pt_out) (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env)
        (p' :timeC->pc_out) (tp' :timeC) .
      PCSet_Correct s' e' p' ==>
        NTH_TIME_TRUE 0 (ale_sig_pb e') 0 tp' ==>
          PT_Exec pti s e p t ==>
            Rst_Slave pti e t e' ==>
              PStateAbs pti s e p t s' e' p' tp' ==>
                (t' <= tp') ==>
                   ~P_rqtS (s' t')"),
     INDUCT_TAC
     THENL [
        % Subgoal 1: (t' Base Case) %
        REWRITE_TAC [PStateAbs]
        THEN REDUCE_TAC
        THEN REPEAT STRIP_TAC
        THEN RES_TAC
     ;
        % Subgoal 2: (t' Induction Step) %
        REPEAT STRIP_TAC
        THEN ASSUME_TAC (SPEC "t':timeC" LESS_EQ_SUC_REFL)
        THEN IMP_RES_TAC LESS_EQ_TRANS
        THEN IMP_RES_TAC ALE_SIG_PB_INIT_FALSE
        THEN IMP_RES_TAC OR_LESS
        THEN RES_TAC
        THEN UNDISCH_TAC "P_rqtS(s'(SUC t'))"
        THEN IMP_RES_TAC P_rqt_ISO
        THEN ASM_REWRITE_TAC [ADD1;COND_TRUE_TRUE;COND_FALSE_CHOICES]
     ]
    );;

let ALE_SIG_IB_FALSE_UPTO_FIRST = TAC_PROOF
    (([],
     "! (t' :timeC) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)        (s :timeT-
>pt_state) (e :timeT->pt_env) (p :timeT->pt_out) (t :timeT)
        (pti :PTI) (tp' :timeC) .
       PCSet_Correct s' e' p' ==>
         NTH_TIME_TRUE 0 (ale_sig_pb e') 0 tp' ==>
           tp' > 0 ==>
             PT_Exec pti s e p t ==>
               PStateAbs pti s e p t s' e' p' tp' ==>
                 Rst_Slave pti e t e' ==>
                   IBA_PMaster pti e p t e' p' ==>
```

```
                    STABLE_FALSE (ale_sig_ib p') (0,tp'-1)"),
     REPEAT STRIP_TAC
     THEN IMP_RES_TAC P_RQT_UPTO_FIRST
     THEN UNDISCH_TAC "NTH_TIME_TRUE 0(ale_sig_pb e')0 tp'"
     THEN REWRITE_TAC [NTH_TIME_TRUE;STABLE_FALSE_THEN_TRUE;STABLE_FALSE;
                       ale_sig_pb;ale_sig_ib;BSel;ZERO_LESS_EQ]
     THEN BETA_TAC
     THEN REPEAT STRIP_TAC
     THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
     THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LE_PRE_IMP_LT)
     THEN IMP_RES_TAC LT_IMP_LE
     THEN SPEC_ASSUM_TAC ("!t'. t' <= tp' ==> ~P_rqtS(s' t')","t':timeC")
     THEN SPEC_ASSUM_TAC
            ("!t. t < tp' ==> ~(~SND(L_ads_E(e' t)) /\ SND(L_den_E(e' t)))",
             "t':timeC")
     THEN RES_TAC
     THENL [
        % Subgoal 1: "SND(I_male_O(p' t')) = LO" %
        UNDISCH_TAC "SND(I_male_O((p':timeC->pc_out) t')) = LO"
        THEN IMP_RES_TAC I_male_ISO
        THEN ASM_REWRITE_TAC [COND_FALSE_TRUE;COND_FALSE_CHOICES;WIRE]
        THEN ASM_CASES_MATCH_RHS_TAC "PH"
        THEN ASM_REWRITE_TAC [prove_constructors_distinct wire;
                              SYM_RULE (prove_constructors_distinct wire)]
      ;
        % Subgoal 2: "SND(I_rale_O(p' t')) = LO" %
        UNDISCH_TAC "SND(I_rale_O((p':timeC->pc_out) t')) = LO"
        THEN IMP_RES_TAC I_rale_ISO
        THEN ASM_REWRITE_TAC [COND_FALSE_TRUE;COND_FALSE_CHOICES;WIRE]
        THEN ASM_CASES_MATCH_RHS_TAC "PH"
        THEN ASM_REWRITE_TAC [prove_constructors_distinct wire;
                              SYM_RULE (prove_constructors_distinct wire)]
     ;
        % Subgoal 3: "~SND(I_cale_O(p' t'))" %
        UNDISCH_TAC "~SND(I_cale_O((p':timeC->pc_out) t'))"
        THEN IMP_RES_TAC IBA_READY_ASSUMPS
        THEN NRULE_ASSUM_TAC
               ("!u':timeC. BSel(I_crqt_O(p' u')) ==> BSel(I_cgnt_E(e' u'))",
                ((REWRITE_RULE [BSel]) o (SPEC "t':timeC")))
        THEN UNDISCH_TAC "SND(I_crqt_O(p' (t':timeC))) ==> SND(I_cgnt_E(e' t'))"
        THEN IMP_RES_TAC I_crqt_ISO
        THEN ASM_REWRITE_TAC[COND_TRUE_TRUE;COND_FALSE_CHOICES]
        THEN DISCH_TAC
        THEN IMP_RES_TAC I_cale_ISO
        THEN ASM_REWRITE_TAC[COND_FALSE_CHOICES;COND_FALSE_TRUE]
        THEN ASM_REWRITE_TAC[]
     ]
    );;

let EXTRACT_ADDR = TAC_PROOF
   (([],
    "SUBARRAY
      (MALTER
       (MALTER
        (ALTER
         (ALTER
          (MALTER
           ARBN
           (31,28)
           ((~P_rqtS((s':timeC->pc_state) ti')) => FST(L_be_E(e' ti'))
                                                 | P_be_S(s' ti')))
          27
          ((~P_rqtS(s' ti')) => FST(L_wrE(e' ti')) | P_wrS(s' ti')))
         26
         F)
        (25,24)
        (SUBARRAY
         ((~P_rqtS(s' ti')) =>
          SUBARRAY(FST(L_ad_inE(e' ti')))(25,0) |
          P_addrS(s' ti'))
         (1,0)))
       (23,0)
```

```
          (SUBARRAY
           ((~P_rqtS(s' ti')) =>
            SUBARRAY(FST(L_ad_inE(e' ti')))(25,0) |
            P_addrS(s' ti'))
           (25,2)))
         (23,0)
         =
         SUBARRAY
           ((~P_rqtS(s' ti')) =>
            SUBARRAY(FST(L_ad_inE(e' ti')))(25,0) |
            P_addrS(s' ti'))
           (25,2)"),
      CONV_TAC (ONCE_DEPTH_CONV FUN_EQ_CONV)
      THEN GEN_TAC
      THEN REWRITE_TAC [ALTER_THM;MALTER_THM;SUBARRAY_THM;ADD_CLAUSES;ZERO_LESS_EQ;
                        COND_TRUE_TRUE]
      THEN ASSUME_TAC (SPEC "n:num" ZERO_LESS_EQ)
      THEN ASSUME_TAC (SPEC "2" ZERO_LESS_EQ)
      THEN IMP_RES_TAC (SPECL ["n:num";"0";"2"] ASSOC_SUB_ADD1)
      THEN ASM_REWRITE_TAC [SYM_RULE (SPECL ["n:num";"23";"2"]
                                      LESS_EQ_MONO_ADD_EQ)]
      THEN REDUCE_TAC
      THEN REWRITE_TAC [COND_TRUE_TRUE]
     );;

let PRE_EXEC_PREC = TAC_PROOF
     (([],
      "! (t :timeT) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
         (pti :PTI) .
       PT_PreC pti s e p (SUC t) ==>
          (?pti0. PT_Exec pti0 s e p t /\ PT_PreC pti0 s e p t)"),
      REWRITE_TAC [PT_PreC]
      THEN REPEAT STRIP_TAC
      THENL [
         EXISTS_TAC "PT_Write"
         THEN ASM_REWRITE_TAC[]
      ;
         EXISTS_TAC "PT_Read"
         THEN ASM_REWRITE_TAC[]
      ]
     );;

let PREC = TAC_PROOF
     (([],
      "! (t :timeT) (pti :PTI)
         (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out) .
       PT_PreC pti s e p t ==>
         ~(PT_fsm_stateS (s t) = PD) /\
         ~PT_rqtS (s t)"),
      INDUCT_TAC
      THEN REWRITE_TAC [PT_PreC]
      THEN REPEAT STRIP_TAC
      THEN RES_TAC
     );;

let ALE_SIG_IB_TRUE_ON_TP' = TAC_PROOF
     (([],
      "! (t :timeT) (pti :PTI)
         (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
         (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
         (tp' :timeC) .
     PCSet_Correct s' e' p' ==>
       NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
         PT_Exec pti s e p t ==>
           PT_PreC pti s e p t ==>
             PStateAbs pti s e p t s' e' p' tp' ==>
               Rst_Slave pti e t e' ==>
                 New_State_Is_PA s' e' tp' ==>
                   ~ELEMENT (FST(L_ad_inE (e' tp'))) 31 ==>
                     ale_sig_ib p' tp'"),
      REWRITE_TAC [New_State_Is_PA;ale_sig_ib;BSel]
      THEN BETA_TAC
      THEN REPEAT STRIP_TAC
      THENL [
         % Subgoal 1: "SND(I_hlda_O(p' tp'))" %
         IMP_RES_TAC I_hlda_ISO
         THEN ASM_REWRITE_TAC
                [SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
      ;
         % Subgoal 2: "(SND(I_male_O(p' tp')) = LO) \/
                       (SND(I_rale_O(p' tp')) = LO) \/
                       ~SND(I_cale_O(p' tp'))" %
         IMP_RES_TAC PREC
         THEN IMP_RES_TAC RST_FALSE
         THEN IMP_RES_TAC NTH_ALE_SIG_PB_TRUE
         THEN SPEC_ASSUM_TAC ("!u'. SND(RstE(e' (u':timeC))) = F","tp':timeC")
         THEN ASM_REWRITE_ASSUM_TAC
                ("PStateAbs pti s e p t s' e' p' tp'",[PStateAbs])
         THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
         THEN IMP_RES_TAC I_male_ISO
         THEN IMP_RES_TAC I_rale_ISO
         THEN ASSUME_TAC (SPEC "25" LESS_EQ_REFL)
         THEN ASM_CASES_TAC "tp'>0"
         THENL [
            % Subgoal 2.1: "tp'>0" %
            RES_TAC
            THEN IMP_RES_TAC
                   (ISPEC "FST(L_ad_inE(e' (tp':timeC)))" SUB_SUBARRAY)
            THEN ASM_REWRITE_TAC
                   [WIRE;SYM_RULE(prove_constructors_distinct pfsm_ty_Axiom);
                    prove_constructors_distinct pfsm_ty_Axiom]
            THEN ASM_CASES_TAC
                   "SUBARRAY(FST(L_ad_inE(e' (tp':timeC))))(25,24) = WORDN 1 3"
         ;
            % Subgoal 2.2: "~tp'>0" %
            REWRITE_ASSUM_TAC ("~tp' > 0",[SYM_RULE NOT_EQ_ZERO])
            THEN IMP_RES_TAC
                   (ISPEC "FST(L_ad_inE(e' 0))" SUB_SUBARRAY)
            THEN ASM_REWRITE_TAC
                   [WIRE;SYM_RULE(prove_constructors_distinct pfsm_ty_Axiom);
                    prove_constructors_distinct pfsm_ty_Axiom]
            THEN ASM_CASES_TAC
                   "SUBARRAY(FST(L_ad_inE(e' 0)))(25,24) = WORDN 1 3"
         ]
         THEN ASM_REWRITE_TAC[prove_constructors_distinct wire]
      ]
     );;

let P_RQT_PREVENTS_NEW_STATE_PD = TAC_PROOF
   (([],
     "! (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
        (t' :timeC) .
      PCSet_Correct s' e' p' ==>
        ~(P_fsm_stateS (s' t') = PD) ==>
          ~P_rqtS (s' t') ==>
            (t' > 0) ==>
              ~New_State_Is_PD s' e' t'"),
      REWRITE_TAC [SYM_RULE ONE_LESS_EQ]
      THEN REPEAT STRIP_TAC
      THEN UNDISCH_TAC "New_State_Is_PD s' e' t'"
      THEN REWRITE_TAC [New_State_Is_PD]
      THEN ASM_CASES_TAC "P_fsm_rstS(s' (t':timeC))"
      THEN IMP_RES_TAC NOT_PD
      THEN ASM_REWRITE_TAC [prove_constructors_distinct pfsm_ty_Axiom;
                            SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
      THENL [
         IMP_RES_TAC SUB_ADD
         THEN IMP_RES_TAC P_fsm_mrqt_ISO
         THEN SPEC_UNDISCH_MATCH_LHS_TAC ("P_fsm_mrqtS(s'(t + 1))","t'-1")
         THEN IMP_RES_TAC P_fsm_crqt_ISO
         THEN SPEC_UNDISCH_MATCH_LHS_TAC ("P_fsm_crqt_S(s'(t + 1))","t'-1")
         THEN IMP_RES_TAC (SYM_RULE P_rqt_ISO)
         THEN SPEC_UNDISCH_MATCH_RHS_TAC ("P_rqtS(s'(t + 1))","t'-1")
         THEN ASM_REWRITE_TAC[]
         THEN DISCH_TAC
         THEN ASM_REWRITE_TAC[]
         THEN DISCH_TAC
         THEN DISCH_TAC
         THEN ASM_REWRITE_TAC[]
         THEN ASM_CASES_TAC "(~P_fsm_hold_S(s' (t':timeC)) /\ P_fsm_lock_S(s' t'))"       THEN ASM_-
REWRITE_TAC [prove_constructors_distinct pfsm_ty_Axiom]
      ;
         ASM_CASES_TAC "P_fsm_hold_S(s' (t':timeC))"
         THEN ASM_REWRITE_TAC [prove_constructors_distinct pfsm_ty_Axiom]
      ]
    );;

let NOT_IBUS_ALE_IMP_NOT_FSM_RQT = TAC_PROOF
    (([],
     "! (t' :timeC) (pti :PTI)
        (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out) (t :timeT)
        (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out).
      PCSet_Correct s' e' p' ==>
        PT_Exec pti s e p t ==>
          IBA_PMaster pti e p t e' p' ==>
            New_State_Is_PA s' e' t' ==>
              ~ale_sig_ib p' t' ==>
                ~(P_fsm_mrqtS(s'(t' + 1)) \/
                  ~P_fsm_crqt_S(s'(t' + 1)) /\ ~P_fsm_cgnt_S(s'(t' + 1)))"),
     REWRITE_TAC [ale_sig_ib;BSel]
     THEN BETA_TAC
     THEN REWRITE_TAC [DE_MORGAN_THM]
     THEN REPEAT STRIP_TAC
     THENL [
        % Subgoal 1: [ "~SND(I_hlda_O(p' t'))" ]
                     [ "P_fsm_mrqtS(s'(t' + 1))" ] %
        REWRITE_ASSUM_TAC ("New_State_Is_PA s' e' t'",[New_State_Is_PA])
        THEN UNDISCH_TAC "~SND(I_hlda_O(p' (t':timeC)))"
        THEN IMP_RES_TAC I_hlda_ISO
        THEN ASM_REWRITE_TAC
                [SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
     ;
        % Subgoal 2: "P_fsm_crqt_S(s'(t' + 1)) \/ P_fsm_cgnt_S(s'(t' + 1))"
                     [ "~SND(I_hlda_O(p' t'))" ] %
        REWRITE_ASSUM_TAC ("New_State_Is_PA s' e' t'",[New_State_Is_PA])
        THEN UNDISCH_TAC "~SND(I_hlda_O(p' (t':timeC)))"
        THEN IMP_RES_TAC I_hlda_ISO
        THEN ASM_REWRITE_TAC
                [SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
     ;
        % Subgoal 3: [ "~(SND(I_male_O(p' t')) = LO)" ]
                     [ "~(SND(I_rale_O(p' t')) = LO)" ]
                     [ "SND(I_cale_O(p' t'))" ]
                     [ "P_fsm_mrqtS(s'(t' + 1))" ] %
        UNDISCH_TAC "P_fsm_mrqtS(s'(t' + 1))"
        THEN IMP_RES_TAC P_fsm_mrqt_ISO
        THEN REWRITE_ASSUM_TAC ("New_State_Is_PA s' e' t'",[New_State_Is_PA])
        THEN ASM_REWRITE_TAC [prove_constructors_distinct pfsm_ty_Axiom]
        THEN STRIP_TAC
        THEN UNDISCH_TAC "~(SND(I_male_O(p' (t':timeC))) = LO)"
        THEN IMP_RES_TAC I_male_ISO
        THEN ASM_REWRITE_TAC
                [SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom);
                 prove_constructors_distinct pfsm_ty_Axiom;WIRE]
        THEN COND_CASES_TAC
        THEN UNDISCH_TAC "~(SND(I_rale_O(p' (t':timeC))) = LO)"
        THEN IMP_RES_TAC I_rale_ISO
        THEN ASM_REWRITE_TAC
                [SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom);
                 prove_constructors_distinct pfsm_ty_Axiom;WIRE]
     ;
        % Subgoal 4: "P_fsm_crqt_S(s'(t' + 1)) \/ P_fsm_cgnt_S(s'(t' + 1))"
                     [ "~(SND(I_male_O(p' t')) = LO)" ]
                     [ "~(SND(I_rale_O(p' t')) = LO)" ]
                     [ "SND(I_cale_O(p' t'))" ] %
        UNDISCH_TAC "SND(I_cale_O(p' (t':timeC)))"
        THEN IMP_RES_TAC I_cale_ISO
        THEN REWRITE_ASSUM_TAC ("New_State_Is_PA s' e' t'",[New_State_Is_PA])
        THEN ASM_REWRITE_TAC[]
        THEN IMP_RES_TAC (REWRITE_RULE [BSel] IBA_READY_ASSUMPS)
        THEN ASM_CASES_TAC "~SND(I_cgnt_E(e' (t':timeC))) /\ SND(I_hold_E(e' t'))"
        THEN ASM_REWRITE_TAC [DE_MORGAN_THM]
        THEN REPEAT STRIP_TAC
        THEN NRULE_ASSUM_TAC
                ("!u'. ~SND(I_cgnt_E(e' (u':timeC))) ==> SND(I_hold_E(e' u'))",
                 ((REWRITE_RULE []) o CONTRAPOS o (SPEC "t':timeC")))
        THEN RES_TAC
        THEN IMP_RES_TAC P_fsm_cgnt_ISO
        THEN ASM_REWRITE_TAC[]
     ]
    );;

let NOT_IBUS_ALE_PREVENTS_NEW_STATE_PD = TAC_PROOF
    (([],
     "! (t' :timeC) (t :timeT) (pti :PTI)
        (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
        (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
        (tp' :timeC) .
      PCSet_Correct s' e' p' ==>
        PT_Exec pti s e p t ==>
          IBA_PMaster pti e p t e' p' ==>
            ~New_State_Is_PD s' e' t' ==>
              ~ale_sig_ib p' t' ==>
                ~New_State_Is_PD s' e' (t'+1)"),
     REPEAT STRIP_TAC
     THEN IMP_RES_TAC NEXT_STATE_NOT_PD
     THEN IMP_RES_TAC NOT_IBUS_ALE_IMP_NOT_FSM_RQT
     THEN UNDISCH_TAC "New_State_Is_PD s' e'(t' + 1)"
     THEN REWRITE_TAC [New_State_Is_PD]
     THEN IMP_RES_TAC P_fsm_state_ISO
     THEN SPEC_UNDISCH_MATCH_LHS_TAC ("P_fsm_stateS(s'(t + 1))","t':timeC")
     THEN ASM_REWRITE_TAC[]
     THEN DISCH_TAC
     THEN ASM_REWRITE_TAC[SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
     THENL [
        % Subgoal 1: [ "new state = PA" ] %
        REWRITE_ASSUM_TAC ("New_State_Is_PA s' e' t'",[New_State_Is_PA])
     ;
        % Subgoal 2: [ "new state = PH" ]%
        REWRITE_ASSUM_TAC ("New_State_Is_PH s' e' t'",[New_State_Is_PH])
     ]
     THEN COND_CASES_TAC
     THEN ASM_REWRITE_TAC
              [SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom);
               prove_constructors_distinct pfsm_ty_Axiom]
     THEN COND_CASES_TAC
     THEN ASM_REWRITE_TAC
              [SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom);
               prove_constructors_distinct pfsm_ty_Axiom]
     THEN COND_CASES_TAC
     THEN ASM_REWRITE_TAC
              [SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom);
               prove_constructors_distinct pfsm_ty_Axiom]
    );;

let SUB_STABLE_FALSE = TAC_PROOF
    (([],
     "! (f :time->bool) (t1 t2 t3 :time) .
      STABLE_FALSE f (t1,t2) ==>
        (t3 <= t2) ==>
          (t1 <= t3) ==>
            STABLE_FALSE f (t1,t3)"),
     REWRITE_TAC [STABLE_FALSE]
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
     THEN SPEC_ASSUM_TAC ("!t. t1 <= t /\ t <= t2 ==> ~f t","t:timeC")
     THEN IMP_RES_TAC LESS_EQ_TRANS
     THEN RES_TAC
    );;

let IBUS_ALE_TRUE_IMP_NEW_STATE_PA = TAC_PROOF
   (([],
    "! (t' :timeC) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out).
     PCSet_Correct s' e' p' ==>
       ale_sig_ib p' t' ==>
         New_State_Is_PA s' e' t'"),
    REWRITE_TAC [ale_sig_ib;BSel;New_State_Is_PA]
    THEN BETA_TAC
    THEN REPEAT GEN_TAC
    THEN DISCH_TAC
    THEN IMP_RES_TAC I_male_ISO
    THEN IMP_RES_TAC I_rale_ISO
    THEN IMP_RES_TAC I_cale_ISO
    THEN ASM_CASES_MATCH_RHS_TAC "PA"
    THEN IMP_RES_TAC NOT_PA
    THEN ASM_REWRITE_TAC
           [WIRE;SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom);
            prove_constructors_distinct wire;
            SYM_RULE (prove_constructors_distinct wire)]
   );;

let OFFSET_IBUS_ALE_FALSE_PREVENTS_NEW_STATE_PD = TAC_PROOF
   (([],
    "! (u' :timeC) (t :timeT) (pti :PTI)
       (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' :timeC) (ti'' :timeC) .
     PCSet_Correct s' e' p' ==>
       NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
         (tp' > 0) ==>
           PT_Exec pti s e p t ==>
             PT_PreC pti s e p t ==>
               PStateAbs pti s e p t s' e' p' tp' ==>
                 Rst_Slave pti e t e' ==>
                   IBA_PMaster pti e p t e' p' ==>
                     STABLE_FALSE (ale_sig_ib p') (tp',(tp'+u')) ==>
                       ((tp'+u') <= ti'') ==>
                         ~New_State_Is_PD s' e' (tp'+u')"),
    REWRITE_TAC [PStateAbs]
    THEN INDUCT_TAC
    THEN REWRITE_TAC [ADD_CLAUSES]
    THEN REPEAT STRIP_TAC
    THENL [
       % Subgoal 1: (Base Case) %
       UNDISCH_TAC "New_State_Is_PD s' e' tp'"
       THEN IMP_RES_TAC PREC
       THEN RES_TAC
       THEN ASM_REWRITE_ASSUM_TAC
               ("P_rqtS(s' (tp':timeC)) = PT_rqtS(s (t:timeT))",[])
       THEN IMP_RES_TAC (SPEC "PT_fsm_stateS(s (t:timeT))" NOT_PD)
       THEN ASM_REWRITE_ASSUM_TAC
               ("P_fsm_stateS(s' (tp':timeC)) = PT_fsm_stateS(s(t:timeT))",[])
       THEN IMP_RES_TAC PA_IMP_NOT_PD
       THEN IMP_RES_TAC PH_IMP_NOT_PD
       THEN IMP_RES_TAC P_RQT_PREVENTS_NEW_STATE_PD
    ;
       % Subgoal 2: (Induction Step) %
       ASSUME_TAC (SPEC "tp'+u'" LESS_EQ_SUC_REFL)
       THEN IMP_RES_TAC LESS_EQ_TRANS
       THEN ASSUME_TAC (SPECL ["tp':timeC";"u':timeC"] LESS_EQ_ADD)
       THEN IMP_RES_TAC SUB_STABLE_FALSE
       THEN RES_TAC
       THEN NRULE_ASSUM_TAC
               ("STABLE_FALSE(ale_sig_ib p')(tp',tp' + u')",
                (BETA_RULE o (REWRITE_RULE [STABLE_FALSE])))
       THEN ASSUME_TAC (SPEC "tp'+u'" LESS_EQ_REFL)
       THEN RES_TAC
       THEN IMP_RES_TAC
              (REWRITE_RULE [SYM_RULE ADD1] NOT_IBUS_ALE_PREVENTS_NEW_STATE_PD)
    ]
   );;

let IBUS_ALE_FALSE_PREVENTS_NEW_STATE_PD = TAC_PROOF
   (([],
    "! (t' :timeC) (t :timeT) (pti :PTI)
       (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' :timeC) (ti'' :timeC) .
     PCSet_Correct s' e' p' ==>
       NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
         (tp' > 0) ==>
           PT_Exec pti s e p t ==>
             PT_PreC pti s e p t ==>
               PStateAbs pti s e p t s' e' p' tp' ==>
                 Rst_Slave pti e t e' ==>
                   IBA_PMaster pti e p t e' p' ==>
                     STABLE_FALSE (ale_sig_ib p') (tp',ti'') ==>
                       (tp' <= t') ==>
                         (t' <= ti'') ==>
                           ~New_State_Is_PD s' e' t'"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC (SPEC "t'-tp'" OFFSET_IBUS_ALE_FALSE_PREVENTS_NEW_STATE_PD)
    THEN SPECL_ASSUM_TAC
            ("!t' tp'''.
              STABLE_FALSE(ale_sig_ib p')(tp',tp' + (t' - tp''')) ==>
                (!ti''.  (tp' + (t' - tp''')) <= ti'' ==>
                         ~New_State_Is_PD s' e'(tp' + (t' - tp''')))",
             ["t':timeC";"tp':timeC"])
    THEN IMP_RES_TAC
             (SPECL ["t':timeC";"tp':timeC"]
                    (PURE_ONCE_REWRITE_RULE [ADD_SYM] SUB_ADD))
    THEN ASM_REWRITE_ASSUM_TAC
             ("STABLE_FALSE(ale_sig_ib p')(tp',tp' + (t' - tp')) ==>
               (!ti''.  (tp' + (t' - tp')) <= ti'' ==>
                        ~New_State_Is_PD s' e'(tp' + (t' - tp')))",[])
    THEN IMP_RES_TAC SUB_STABLE_FALSE
    THEN RES_TAC
    THEN SPEC_ASSUM_TAC
             ("!ti''.  (tp' + (t' - tp')) <= ti'' ==>
                       ~New_State_Is_PD s' e'(tp' + (t' - tp'))","ti'':timeC")
    THEN ASM_REWRITE_ASSUM_TAC
             ("(tp' + (t' - tp')) <= ti'' ==>
                  ~New_State_Is_PD s' e'(tp' + (t' - tp'))",[])
   );;

let OFFSET_NEW_STATE_PD_FALSE_FROM_TP'_TO_TI' = TAC_PROOF
   (([],
    "! (u' :timeC) (t :timeT) (pti :PTI)
       (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' :timeC) (ti' :timeC) .
     PCSet_Correct s' e' p' ==>
       NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
         (tp' > 0) ==>
           PT_Exec pti s e p t ==>
             PT_PreC pti s e p t ==>
               PStateAbs pti s e p t s' e' p' tp' ==>
                 Rst_Slave pti e t e' ==>
                   IBA_PMaster pti e p t e' p' ==>
                     STABLE_FALSE (ale_sig_ib p') (tp',ti'-1) ==>
                       ((tp'+u') <= ti') ==>
                         ~New_State_Is_PD s' e' (tp'+u')"),
    REPEAT STRIP_TAC
    THEN ASM_CASES_TAC "ale_sig_ib p' (tp'+u')"
    THENL [
       % Subgoal 1: [ "ale_sig_ib p'(tp' + u')" ] %
       IMP_RES_TAC IBUS_ALE_TRUE_IMP_NEW_STATE_PA
       THEN REWRITE_ASSUM_TAC("New_State_Is_PD s' e' (tp'+u')",[New_State_Is_PD])
       THEN REWRITE_ASSUM_TAC("New_State_Is_PA s' e' (tp'+u')",[New_State_Is_PA])
       THEN IMP_RES_TAC PA_IMP_NOT_PD
    ;
       % Subgoal 2: [ "~ale_sig_ib p'(tp' + u')" ] %
```

```
      REWRITE_ASSUM_TAC ("(tp'+u') <= ti'",[LESS_OR_EQ])
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THENL [
         % Subgoal 2.1: [ "(tp'+u') < ti'" ] %
         SUBGOAL_THEN "STABLE_FALSE (ale_sig_ib p') (tp',tp'+u')" ASSUME_TAC
         THENL [
            % Subgoal 2.1.1: (New subgoal) %
            UNDISCH_TAC "STABLE_FALSE(ale_sig_ib p')(tp',ti' - 1)"
            THEN ASM_REWRITE_TAC [STABLE_FALSE]
            THEN REPEAT STRIP_TAC
            THEN REWRITE_TAC [LESS_EQ_ADD]
            THEN IMP_RES_TAC SUB_LESS_OR
            THEN IMP_RES_TAC LESS_EQ_TRANS
            THEN RES_TAC
         ;
            % Subgoal 2.1.2 %
            ASSUME_TAC (SPEC "tp'+u'" LESS_EQ_REFL)
            THEN IMP_RES_TAC
                  (SPEC "tp':timeC" OFFSET_IBUS_ALE_FALSE_PREVENTS_NEW_STATE_PD)
         ]
      ;
         % Subgoal 2.2: [ "t' = ti'" ] %
         SUBGOAL_THEN "STABLE_FALSE (ale_sig_ib p') (tp',tp'+u')" ASSUME_TAC
         THENL [
            % Subgoal 2.2.1: (New subgoal) %
            UNDISCH_TAC "STABLE_FALSE(ale_sig_ib p')(tp',ti' - 1)"
            THEN ASM_REWRITE_ASSUM_TAC ("~ale_sig_ib p' (tp'+u')",[])
            THEN ASM_REWRITE_TAC[STABLE_FALSE]
            THEN REPEAT STRIP_TAC
            THENL [
               ASSUME_TAC (SPECL ["ti':timeC";"1"] SUB_LESS_EQ)
               THEN IMP_RES_TAC LESS_EQ_TRANS
            ;
               REWRITE_ASSUM_TAC ("t' <= ti'",[LESS_OR_EQ])
               THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
               THENL [
                 IMP_RES_TAC SUB_LESS_OR
                 THEN RES_TAC
               ;
                 UNDISCH_TAC "ale_sig_ib p' t'"
                 THEN ASM_REWRITE_TAC[]
               ]
            ]
         ;
            % Subgoal 2.2.2 %
            ASSUME_TAC (SPEC "tp'+u'" LESS_EQ_REFL)
            THEN ASSUME_TAC (SPECL ["tp':timeC";"u':timeC"] LESS_EQ_ADD)
            THEN IMP_RES_TAC
                   (SPEC "tp'+u'" IBUS_ALE_FALSE_PREVENTS_NEW_STATE_PD)
         ]
      ]
   ]
  );;

let NEW_STATE_PD_FALSE_FROM_TP'_TO_TI' = TAC_PROOF
   (([],
    "! (t' :timeC) (t :timeT) (pti :PTI)
       (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' :timeC) (ti' :timeC) .
     PCSet_Correct s' e' p' ==>
      NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
        (tp' > 0) ==>
          PT_Exec pti s e p t ==>
            PT_PreC pti s e p t ==>
              PStateAbs pti s e p t s' e' p' tp' ==>
                Rst_Slave pti e t e' ==>
                  IBA_PMaster pti e p t e' p' ==>
                    STABLE_FALSE (ale_sig_ib p') (tp',ti'-1) ==>
                      (tp' <= t') ==>
                        (t' <= ti') ==>
                          ~New_State_Is_PD s' e' t'"),
    REPEAT STRIP_TAC
    THEN ASM_CASES_TAC "ale_sig_ib p' t'"
    THENL [
       % Subgoal 1: [ "ale_sig_ib p' t'" ] %
       IMP_RES_TAC IBUS_ALE_TRUE_IMP_NEW_STATE_PA
       THEN REWRITE_ASSUM_TAC ("New_State_Is_PD s' e' t'",[New_State_Is_PD])
       THEN REWRITE_ASSUM_TAC ("New_State_Is_PA s' e' t'",[New_State_Is_PA])
       THEN IMP_RES_TAC PA_IMP_NOT_PD
    ;
       % Subgoal 2: [ "~ale_sig_ib p' t'" ] %
       REWRITE_ASSUM_TAC ("t' <= ti'",[LESS_OR_EQ])
       THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
       THENL [
          % Subgoal 2.1: [ "t' < ti'" ] %
          SUBGOAL_THEN "STABLE_FALSE (ale_sig_ib p') (tp',t')" ASSUME_TAC
          THENL [
             % Subgoal 2.1.1: (New subgoal) %
             UNDISCH_TAC "STABLE_FALSE(ale_sig_ib p')(tp',ti' - 1)"
             THEN ASM_REWRITE_TAC [STABLE_FALSE]
             THEN REPEAT STRIP_TAC
             THEN IMP_RES_TAC SUB_LESS_OR
             THEN IMP_RES_TAC LESS_EQ_TRANS
             THEN RES_TAC
          ;
             % Subgoal 2.1.2 %
             ASSUME_TAC (SPEC "t':timeC" LESS_EQ_REFL)
             THEN IMP_RES_TAC
                    (SPEC "t':timeC" IBUS_ALE_FALSE_PREVENTS_NEW_STATE_PD)
          ]
       ;
          % Subgoal 2.2: [ "t' = ti'" ] %
          SUBGOAL_THEN "STABLE_FALSE (ale_sig_ib p') (tp',t')" ASSUME_TAC
          THENL [
             % Subgoal 2.2.1: (New subgoal) %
             UNDISCH_TAC "STABLE_FALSE(ale_sig_ib p')(tp',ti' - 1)"
             THEN ASM_REWRITE_ASSUM_TAC ("~ale_sig_ib p' t'",[])
             THEN ASM_REWRITE_TAC[STABLE_FALSE]
             THEN REPEAT STRIP_TAC
             THENL [
                ASM_REWRITE_ASSUM_TAC ("tp' <= t'",[])
             ;
                REWRITE_ASSUM_TAC ("t'' <= ti'",[LESS_OR_EQ])
                THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
                THENL [
                  IMP_RES_TAC SUB_LESS_OR
                  THEN RES_TAC
                ;
                  UNDISCH_TAC "ale_sig_ib p' t''"
                  THEN ASM_REWRITE_TAC[]
                ]
             ]
          ;
             % Subgoal 2.2.2 %
             ASSUME_TAC (SPEC "t':timeC" LESS_EQ_REFL)
             THEN IMP_RES_TAC
                    (SPEC "t':timeC" IBUS_ALE_FALSE_PREVENTS_NEW_STATE_PD)

          ]
       ]
    ]
   );;

let OFFSET_NEW_P_RQT_TRUE_FROM_TP'_TO_TI' = TAC_PROOF
   (([],
    "! (u' :timeC) (t :timeT) (pti :PTI)
       (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' :timeC) (ti' :timeC) .
     PCSet_Correct s' e' p' ==>
       NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
         (tp' > 0) ==>
           PT_Exec pti s e p t ==>
```

```
            PT_PreC pti s e p t ==>
              PStateAbs pti s e p t s' e' p' tp' ==>
                Rst_Slave pti e t e' ==>
                  IBA_PMaster pti e p t e' p' ==>
                    STABLE_FALSE (ale_sig_ib p') (tp',(ti'-1)) ==>
                      ((tp'+u') <= ti') ==>
                        New_P_Rqt_Is_TRUE s' e' (tp'+u')"),
    INDUCT_TAC
    THENL [
      % Subgoal 1: (Base Case) %
      REWRITE_TAC [STABLE_FALSE;STABLE_TRUE;PStateAbs;ADD_CLAUSES]
      THEN BETA_TAC
      THEN REPEAT STRIP_TAC
      THEN ASM_REWRITE_TAC[]
      THEN IMP_RES_TAC NTH_ALE_SIG_PB_TRUE
      THEN IMP_RES_TAC RST_FALSE
      THEN SPEC_ASSUM_TAC ("!u':timeC. SND(RstE(e' u')) = F","tp':timeC")
      THEN RES_TAC
      THEN IMP_RES_TAC PREC
      THEN IMP_RES_TAC (SPEC "PT_fsm_stateS(s (t:timeT))" NOT_PD)
      THEN ASM_REWRITE_ASSUM_TAC
              ("P_fsm_stateS(s' (tp':timeC)) = PT_fsm_stateS(s (t:timeT))",[])
      THEN ASM_REWRITE_ASSUM_TAC
              ("P_rqtS(s' (tp':timeC)) = PT_rqtS(s (t:timeT))",[])
      THEN IMP_RES_TAC PA_IMP_NOT_PD
      THEN IMP_RES_TAC PH_IMP_NOT_PD
      THEN IMP_RES_TAC P_RQT_PREVENTS_NEW_STATE_PD
      THEN ASM_REWRITE_TAC [New_P_Rqt_Is_TRUE]
    ;
      % Subgoal 2: (Induction Step) %
      REWRITE_TAC [ADD1;ADD_ASSOC]
      THEN REPEAT STRIP_TAC
      THEN ASSUME_TAC (SPECL ["tp'+u'";"1"] LESS_EQ_ADD)
      THEN ASSUME_TAC (SPECL ["tp':timeC";"u':timeC"] LESS_EQ_ADD)
      THEN IMP_RES_TAC LESS_EQ_TRANS
      THEN IMP_RES_TAC SUB_STABLE_FALSE
      THEN RES_TAC
      THEN REWRITE_ASSUM_TAC
             ("STABLE_FALSE(ale_sig_ib p')(tp',(tp' + u') + 1)",
              [ASSOC_ADD_ADD1])
      THEN REWRITE_ASSUM_TAC
             ("((tp' + u') + 1) <= ti'",[ASSOC_ADD_ADD1])
      THEN IMP_RES_TAC OFFSET_NEW_STATE_PD_FALSE_FROM_TP'_TO_TI'
      THEN REWRITE_ASSUM_TAC
             ("~New_State_Is_PD s' e'(tp' + (u' + 1))",[ADD_ASSOC])
      THEN IMP_RES_TAC RST_FALSE
      THEN SPEC_ASSUM_TAC ("!u'. SND(RstE(e' (u':timeC))) = F","(tp'+u')+1")
      THEN IMP_RES_TAC P_rqt_ISO
      THEN RES_TAC
      THEN SPEC_UNDISCH_MATCH_LHS_TAC ("P_rqtS(s'(t + 1))","tp'+u'")
      THEN REWRITE_ASSUM_TAC
             ("New_P_Rqt_Is_TRUE s' e'(tp' + u')",
              [New_P_Rqt_Is_TRUE;New_State_Is_PD])
      THEN ASM_REWRITE_TAC[]
      THEN DISCH_TAC
      THEN ASM_REWRITE_TAC [New_P_Rqt_Is_TRUE;COND_TRUE_TRUE;COND_TRUE_CHOICES]
    ]
   );;

let NEW_P_RQT_TRUE_FROM_TP'_TO_TI' = TAC_PROOF
  (([],
   "! (t' :timeC) (t :timeT) (pti :PTI)
      (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
      (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
      (tp' :timeC) (ti' :timeC) .
    PCSet_Correct s' e' p' ==>
      NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
        (tp' > 0) ==>
          PT_Exec pti s e p t ==>
            PT_PreC pti s e p t ==>
              PStateAbs pti s e p t s' e' p' tp' ==>
                Rst_Slave pti e t e' ==>
```

```
                    IBA_PMaster pti e p t e' p' ==>
                      STABLE_FALSE (ale_sig_ib p') (tp',ti'-1) ==>
                        (tp' <= t') ==>
                          (t' <= ti') ==>
                            New_P_Rqt_Is_TRUE s' e' t'"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC (SPEC "t'-tp'" OFFSET_NEW_P_RQT_TRUE_FROM_TP'_TO_TI')
    THEN SPECL_ASSUM_TAC
           ("!t' tp'''. (tp' + (t' - tp''')) <= ti' ==>
                        New_P_Rqt_Is_TRUE s' e'(tp' + (t' - tp'''))",
            ["t':timeC";"tp':timeC"])
    THEN IMP_RES_TAC
            (SPECL ["t':timeC";"tp':timeC"]
                   (PURE_ONCE_REWRITE_RULE [ADD_SYM] SUB_ADD))
    THEN ASM_REWRITE_ASSUM_TAC
            ("(tp' + (t' - tp')) <= ti' ==>
                New_P_Rqt_Is_TRUE s' e'(tp' + (t' - tp'))",[])
   );;

let OFFSET_NEW_P_DEST1_STABLE_FROM_TP'_TO_TI' = TAC_PROOF
   (([],
    "! (u' :timeC) (t :timeT) (pti :PTI)
       (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' :timeC) (ti' :timeC) .
     PCSet_Correct s' e' p' ==>
       NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
         (tp' > 0) ==>
           PT_Exec pti s e p t ==>
             PT_PreC pti s e p t ==>
               PStateAbs pti s e p t s' e' p' tp' ==>
                 Rst_Slave pti e t e' ==>
                   IBA_PMaster pti e p t e' p' ==>
                     STABLE_FALSE (ale_sig_ib p') (tp',ti'-1) ==>
                       ((tp'+u') <= ti') ==>
                         (P_dest1S (s' (tp'+u'+1)) =
                            ELEMENT (FST(L_ad_inE (e' tp'))) 31)"),
    INDUCT_TAC
    THENL [
       % Subgoal 1: (Base case) %
       REWRITE_TAC [ADD_CLAUSES;PStateAbs]
       THEN REPEAT STRIP_TAC
       THEN IMP_RES_TAC PREC
       THEN RES_TAC
       THEN IMP_RES_TAC P_dest1_ISO
       THEN ASM_REWRITE_TAC[]
    ;
       % Subgoal 2: (Induction step) %
       REWRITE_TAC [ADD_ASSOC;ADD1]
       THEN REPEAT STRIP_TAC
       THEN ASSUME_TAC (SPECL ["tp':timeC";"u':timeC"] LESS_EQ_ADD)
       THEN ASSUME_TAC (SPECL ["tp'+u'";"1"] LESS_EQ_ADD)
       THEN IMP_RES_TAC SUB_STABLE_FALSE
       THEN IMP_RES_TAC LESS_EQ_TRANS
       THEN RES_TAC
       THEN ASSUME_TAC (SPEC "tp'+u'" LESS_EQ_REFL)
       THEN IMP_RES_TAC NEW_P_RQT_TRUE_FROM_TP'_TO_TI'
       THEN REWRITE_ASSUM_TAC
              ("P_dest1S(s'(tp' + (u' + 1))) = ELEMENT(FST(L_ad_inE(e' tp')))31",
               [ADD_ASSOC])
       THEN IMP_RES_TAC P_rqt_ISO
       THEN SPEC_UNDISCH_MATCH_LHS_TAC
              ("P_rqtS(s'(t + 1))","tp'+u'")
       THEN REWRITE_ASSUM_TAC
              ("New_P_Rqt_Is_TRUE s' e'(tp' + u')",
               [New_P_Rqt_Is_TRUE;New_State_Is_PD])
       THEN ASM_REWRITE_TAC[]
       THEN DISCH_TAC
       THEN IMP_RES_TAC P_dest1_ISO
       THEN ASM_REWRITE_TAC[ASSOC_ADD_ADD1]
    ]
   );;
```

```
let NEW_P_DEST1_STABLE_FROM_TP'_TO_TI' = TAC_PROOF
   (([],
    "! (t' :timeC) (t :timeT) (pti :PTI)
       (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' :timeC) (ti' :timeC) .
     PCSet_Correct s' e' p' ==>
       NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
         (tp' > 0) ==>
           PT_Exec pti s e p t ==>
             PT_PreC pti s e p t ==>
               PStateAbs pti s e p t s' e' p' tp' ==>
                 Rst_Slave pti e t e' ==>
                   IBA_PMaster pti e p t e' p' ==>
                     STABLE_FALSE (ale_sig_ib p') (tp',ti'-1) ==>
                       (tp' <= t') ==>
                         (t' <= ti') ==>
                           (P_dest1S (s' (t'+1)) =
                              ELEMENT (FST(L_ad_inE (e' tp'))) 31)"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC (SPEC "t'-tp'" OFFSET_NEW_P_DEST1_STABLE_FROM_TP'_TO_TI')
    THEN SPECL_ASSUM_TAC
           ("!t' tp'''. (tp' + (t' - tp''')) <= ti' ==>
                        (P_dest1S(s'(tp' + ((t' - tp''') + 1))) =
                          ELEMENT(FST(L_ad_inE(e' tp')))31)",
            ["t':timeC";"tp':timeC"])
    THEN ASSUME_TAC (SPEC "tp':timeC" LESS_EQ_REFL)
    THEN IMP_RES_TAC
           (SYM_RULE (SPECL ["tp':timeC";"tp':timeC";"t':timeC"] ASSOC_SUB_ADD1))
    THEN ASM_REWRITE_ASSUM_TAC
           ("(tp' + (t' - tp')) <= ti' ==>
               (P_dest1S(s'(tp' + ((t' - tp') + 1))) =
                 ELEMENT(FST(L_ad_inE(e' tp')))31)",
            [ADD_CLAUSES;SUB_EQUAL_0])
    THEN ASSUME_TAC
           (SYM_RULE (SPECL ["t'-tp'";"tp':timeC";"1"] ASSOC_ADD_ADD3))
    THEN IMP_RES_TAC (SPECL ["t':timeC";"tp':timeC"] SUB_ADD)
    THEN ASM_REWRITE_ASSUM_TAC
           ("P_dest1S(s'(tp' + ((t' - tp') + 1))) =
             ELEMENT(FST(L_ad_inE(e' tp')))31",[])
   );;

let OFFSET_NEW_P_ADDR_STABLE_FROM_TP'_TO_TI' = TAC_PROOF
   (([],
    "! (u' :timeC) (t :timeT) (pti :PTI)
       (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' :timeC) (ti' :timeC) .
     PCSet_Correct s' e' p' ==>
       NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
         (tp' > 0) ==>
           PT_Exec pti s e p t ==>
             PT_PreC pti s e p t ==>
               PStateAbs pti s e p t s' e' p' tp' ==>
                 Rst_Slave pti e t e' ==>
                   IBA_PMaster pti e p t e' p' ==>
                     STABLE_FALSE (ale_sig_ib p') (tp',ti'-1) ==>
                       ((tp'+u') <= ti') ==>
                         (P_addrS (s' (tp'+u'+1)) =
                            SUBARRAY (FST(L_ad_inE(e' tp'))) (25,0))"),
    INDUCT_TAC
    THENL [
       % Subgoal 1: (Base case) %
       REWRITE_TAC [ADD_CLAUSES;PStateAbs]
       THEN REPEAT STRIP_TAC
       THEN IMP_RES_TAC PREC
       THEN RES_TAC
       THEN IMP_RES_TAC P_addr_ISO
       THEN ASM_REWRITE_TAC[]
    ;
       % Subgoal 2: (Induction step) %
```

```
      REWRITE_TAC [ADD_ASSOC;ADD1]
      THEN REPEAT STRIP_TAC
      THEN ASSUME_TAC (SPECL ["tp':timeC";"u':timeC"] LESS_EQ_ADD)
      THEN ASSUME_TAC (SPECL ["tp'+u'";"1"] LESS_EQ_ADD)
      THEN IMP_RES_TAC SUB_STABLE_FALSE
      THEN IMP_RES_TAC LESS_EQ_TRANS
      THEN RES_TAC
      THEN ASSUME_TAC (SPEC "tp'+u'" LESS_EQ_REFL)
      THEN IMP_RES_TAC NEW_P_RQT_TRUE_FROM_TP'_TO_TI'
      THEN REWRITE_ASSUM_TAC
             ("P_addrS(s'(tp' + (u' + 1))) =
                 SUBARRAY (FST(L_ad_inE(e' tp'))) (25,0)",
              [ADD_ASSOC])
      THEN IMP_RES_TAC P_rqt_ISO
      THEN SPEC_UNDISCH_MATCH_LHS_TAC
             ("P_rqtS(s'(t + 1))","tp'+u'")
      THEN REWRITE_ASSUM_TAC
             ("New_P_Rqt_Is_TRUE s' e'(tp' + u')",
              [New_P_Rqt_Is_TRUE;New_State_Is_PD])
      THEN ASM_REWRITE_TAC[]
      THEN DISCH_TAC
      THEN IMP_RES_TAC P_addr_ISO
      THEN ASM_REWRITE_TAC[ASSOC_ADD_ADD1]
    ]
  );;

let NEW_P_ADDR_STABLE_FROM_TP'_TO_TI' = TAC_PROOF
   (([],
    "! (t' :timeC) (t :timeT) (pti :PTI)
       (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' :timeC) (ti' :timeC) .
     PCSet_Correct s' e' p' ==>
       NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
         (tp' > 0) ==>
           PT_Exec pti s e p t ==>
             PT_PreC pti s e p t ==>
               PStateAbs pti s e p t s' e' p' tp' ==>
                 Rst_Slave pti e t e' ==>
                   IBA_PMaster pti e p t e' p' ==>
                     STABLE_FALSE (ale_sig_ib p') (tp',ti'-1) ==>
                       (tp' <= t') ==>
                         (t' <= ti') ==>
                           (P_addrS (s' (t'+1)) =
                              SUBARRAY (FST(L_ad_inE(e' tp'))) (25,0))"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC (SPEC "t'-tp'" OFFSET_NEW_P_ADDR_STABLE_FROM_TP'_TO_TI')
    THEN SPECL_ASSUM_TAC
           ("!t' tp'''. (tp' + (t' - tp''')) <= ti' ==>
                          (P_addrS(s'(tp' + ((t' - tp''') + 1))) =
                             SUBARRAY(FST(L_ad_inE(e' tp')))(25,0))",
            ["t':timeC";"tp':timeC"])
    THEN ASSUME_TAC (SPEC "tp':timeC" LESS_EQ_REFL)
    THEN IMP_RES_TAC
           (SYM_RULE (SPECL ["tp':timeC";"tp':timeC";"t':timeC"] ASSOC_SUB_ADD1))
    THEN ASM_REWRITE_ASSUM_TAC
           ("(tp' + (t' - tp')) <= ti' ==>
               (P_addrS(s'(tp' + ((t' - tp') + 1))) =
                 SUBARRAY(FST(L_ad_inE(e' tp')))(25,0))",
            [ADD_CLAUSES;SUB_EQUAL_0])
    THEN ASSUME_TAC
           (SYM_RULE (SPECL ["t'-tp'";"tp':timeC";"1"] ASSOC_ADD_ADD3))
    THEN IMP_RES_TAC (SPECL ["t':timeC";"tp':timeC"] SUB_ADD)
    THEN ASM_REWRITE_ASSUM_TAC
          ("P_addrS(s'(tp' + ((t' - tp') + 1))) =
              SUBARRAY(FST(L_ad_inE(e' tp')))(25,0)",[])
  );;

let EVENTUALLY_PA_ON_OR_AFTER_PH = mk_thm
   ([], "! (t' :timeC) (s' :timeC->pc_state) (e' :timeC->pc_env)
           (p' :timeC->pc_out) .
     PCSet_Correct s' e' p' ==>
```

```
          PT_Exec pti s e p t ==>
           IBA_PMaster pti e p t e' p' ==>
            Rst_Slave pti e t e' ==>
             (t' > 0) ==>
              New_State_Is_PH s' e' t' ==>
               (?u':timeC.
                STABLE_FALSE_THEN_TRUE (\v'. New_State_Is_PA s' e' v') (t',u'))");;

%
let EVENTUALLY_PA_ON_OR_AFTER_PH = TAC_PROOF
   (([],
    "! (t' :timeC) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (t :timeT) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (pti :PTI) .
     PCSet_Correct s' e' p' ==>
      PT_Exec pti s e p t ==>
       IBA_PMaster pti e p t e' p' ==>
        Rst_Slave pti e t e' ==>
         (t' > 0) ==>
          New_State_Is_PH s' e' t' ==>
           (?u':timeC.
            STABLE_FALSE_THEN_TRUE (\v'. New_State_Is_PA s' e' v') (t',u'))"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC IBA_READY_ASSUMPS
    THEN SPEC_ASSUM_TAC
           ("!u'. ?v'. STABLE_FALSE_THEN_TRUE(bsig I_hold_E e')(u',v')",
            "t'-1")
    THEN CHOOSE_ASSUM_TAC
           "?v'. STABLE_FALSE_THEN_TRUE(bsig I_hold_E e')(t'-1,v')"
    THEN EXISTS_TAC "v':timeC"
    THEN REWRITE_TAC [STABLE_FALSE_THEN_TRUE]
    THEN BETA_TAC
    THEN NRULE_ASSUM_TAC
           ("STABLE_FALSE_THEN_TRUE(bsig I_hold_E e')(t'-1,v')",
            (BETA_RULE o (REWRITE_RULE [STABLE_FALSE_THEN_TRUE;bsig;BSel])))
    THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
    THEN ASM_REWRITE_TAC[]



    THEN SUBGOAL_THEN
           "!u'. New_State_Is_PH s' e' u' ==>
                   ~SND(I_hold_E (e' u')) ==>
                     (1 <= u') ==>
                       New_State_Is_PH s' e' (u'+1)" ASSUME_TAC
    THENL [
        Subgoal 1: (New Subgoal)
       REPEAT STRIP_TAC
       THEN ASSUME_TAC (GEN_ALL P_fsm_hold_ISO)
       THEN SPECL_ASSUM_TAC
              ("!s e p t. PCSet_Correct s e p ==>
                          (P_fsm_hold_S(s(t + 1)) = SND(I_hold_E(e t)))",
               ["s':timeC->pc_state";"e':timeC->pc_env";"p':timeC->pc_out";
                "u':timeC"])
       THEN RES_TAC
       THEN ASSUME_TAC (SPECL ["u':timeC";"1"] LESS_EQ_ADD)
       THEN IMP_RES_TAC LESS_EQ_TRANS
       THEN IMP_RES_TAC P_FSM_RST_FALSE
       THEN IMP_RES_TAC P_fsm_state_ISO
       THEN SUBGOAL_THEN "P_fsm_stateS (s' (u'+1)) = PH" ASSUME_TAC
       THENL [
          Subgoal 1.1: (New Subgoal)
         REWRITE_ASSUM_TAC ("New_State_Is_PH s' e' u'",[New_State_Is_PH])
         THEN ASM_REWRITE_TAC[]



       THEN SUBGOAL_THEN "~P_fsm_hold_S (s' (u'+1))" ASSUME_TAC
       THENL [
          Subgoal 1.1: (New Subgoal)
          THEN ASM_REWRITE_TAC[]
   );;
%

let EVENTUALLY_PA_AFTER_PH = TAC_PROOF
   (([],
    "! (t' :timeC) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (t :timeT) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (pti :PTI) .
     PCSet_Correct s' e' p' ==>
      PT_Exec pti s e p t ==>
       IBA_PMaster pti e p t e' p' ==>
        Rst_Slave pti e t e' ==>
         (t' > 0) ==>
          New_State_Is_PH s' e' t' ==>
           (?u':timeC.
            (t' < u') /\
            STABLE_FALSE_THEN_TRUE (\v'. New_State_Is_PA s' e' v') (t',u'))"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC EVENTUALLY_PA_ON_OR_AFTER_PH
    THEN EXISTS_TAC "u':timeC"
    THEN ASM_REWRITE_TAC[]
    THEN NRULE_ASSUM_TAC
           ("STABLE_FALSE_THEN_TRUE(\v'. New_State_Is_PA s' e' v')(t',u')",
            (BETA_RULE o (REWRITE_RULE [STABLE_FALSE_THEN_TRUE])))
    THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
    THEN IMP_RES_TAC LESS_OR_EQ
    THEN UNDISCH_TAC "New_State_Is_PH s' e' t'"
    THEN ASM_REWRITE_TAC[New_State_Is_PH]
    THEN REWRITE_ASSUM_TAC ("New_State_Is_PA s' e' u'",[New_State_Is_PA])
    THEN DISCH_TAC
    THEN IMP_RES_TAC PH_IMP_NOT_PA
   );;

let NOT_PTSTATE_IMP_NOT_PCSTATE = TAC_PROOF
   (([],
    "! (x :pfsm_ty) (s :timeT->pt_state) (t :timeT) (s' :timeC->pc_state)
       (t' :timeC) .
     ~(PT_fsm_stateS(s t) = x) ==>
       (P_fsm_stateS(s' t') = PT_fsm_stateS(s t)) ==>
         ~(P_fsm_stateS(s' t') = x)"),
    INDUCT_THEN (prove_induction_thm pfsm_ty_Axiom) ASSUME_TAC
    THEN REPEAT STRIP_TAC
    THENL [
       IMP_RES_TAC NOT_PH
       THEN ASM_REWRITE_ASSUM_TAC
              ("P_fsm_stateS(s' (t':timeC)) = PH",
               [SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)])
    ;
       IMP_RES_TAC NOT_PA
       THEN ASM_REWRITE_ASSUM_TAC
              ("P_fsm_stateS(s' (t':timeC)) = PA",
               [SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom);
                prove_constructors_distinct pfsm_ty_Axiom])
    ;
       IMP_RES_TAC NOT_PD
       THEN ASM_REWRITE_ASSUM_TAC
              ("P_fsm_stateS(s' (t':timeC)) = PD",
               [SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom);
                prove_constructors_distinct pfsm_ty_Axiom])
    ]
   );;

let PREC_TAC =
      IMP_RES_TAC PREC
      THEN REWRITE_ASSUM_TAC
             ("PStateAbs pti s e p t s' e' p' tp'",[PStateAbs])
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN RES_TAC
      THEN ASM_REWRITE_ASSUM_TAC
             ("P_rqtS(s' (tp':timeC)) = PT_rqtS(s (t:timeT))",[])
      THEN IMP_RES_TAC NOT_PTSTATE_IMP_NOT_PCSTATE;;
```

```
let P_DEST1_TRUE_IMP_P_FSM_MRQT_FALSE = TAC_PROOF
   (([],
    "! (t' :timeC) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out).
     PCSet_Correct s' e' p' ==>
       (t' > 0) ==>
         P_dest1S (s' t') ==>
           ~P_fsm_mrqtS (s' t')"),
    REPEAT GEN_TAC
    THEN DISCH_TAC
    THEN DISCH_TAC
    THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
    THEN IMP_RES_TAC (SPECL ["t':timeC";"1"] (SYM_RULE SUB_ADD))
    THEN PURE_ONCE_ASM_REWRITE_TAC[]
    THEN POP_ASSUM (\thm. ALL_TAC) % KEEP THIS %
    THEN DISCH_TAC
    THEN IMP_RES_TAC (SYM_RULE P_dest1_ISO)
    THEN IMP_RES_TAC P_fsm_mrqt_ISO
    THEN ASM_REWRITE_TAC[]
   );;

let ALE_SIG_IB_FALSE_AWAITING_CGNT = TAC_PROOF
   (([],
    "! (t :timeT) (pti :PTI)
       (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' :timeC) .
    PCSet_Correct s' e' p' ==>
     NTH_TIME_TRUE t(ale_sig_pb e')0 tp' ==>
      tp' > 0 ==>
       PT_Exec pti s e p t ==>
        PT_PreC pti s e p t ==>
         PStateAbs pti s e p t s' e' p' tp' ==>
          IBA_PMaster pti e p t e' p' ==>
           Rst_Slave pti e t e' ==>
            STABLE_TRUE_THEN_FALSE (bsig I_cgnt_E e') (tp',ti') ==>
             ELEMENT (FST(L_ad_inE (e' tp'))) 31 ==>
              (tp' < ti') ==>
               STABLE_FALSE (ale_sig_ib p') (tp',ti'-1)"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LT_IMP_LE_PRE)
    THEN ASM_REWRITE_TAC [STABLE_FALSE]
    THEN REPEAT STRIP_TAC
    THEN ASSUME_TAC
           (BETA_RULE
             (SPECL ["ale_sig_ib p'";"t':timeC";"tp':timeC";"ti'-1"]
                    FIRST_EXISTS1))
    THEN RES_TAC
    THEN DELETE_ASSUM_TAC
           "ale_sig_ib p' t' /\ tp' <= t' /\ t' <= (ti' - 1) ==>
            (?u. tp' <= u /\ u <= (ti' - 1) /\
            STABLE_FALSE_THEN_TRUE(ale_sig_ib p')(tp',u))"
    THEN SUBGOAL_THEN "tp' < u" ASSUME_TAC
    THENL [
       % Subgoal 1: (New Subgoal) %
       REWRITE_ASSUM_TAC ("tp' <= u",[LESS_OR_EQ])
       THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
       THEN ASM_REWRITE_TAC[]
       THEN PREC_TAC
       THEN ASM_REWRITE_ASSUM_TAC ("ELEMENT(FST(L_ad_inE(e' (tp':timeC))))31",[])
       THEN NRULE_ASSUM_TAC
              ("STABLE_TRUE_THEN_FALSE(bsig I_cgnt_E e')(tp',ti')",
               (BETA_RULE o (REWRITE_RULE [STABLE_TRUE_THEN_FALSE;bsig;BSel])))
       THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
       THEN ASSUME_TAC (SPEC "tp':timeC" LESS_EQ_REFL)
       THEN SPEC_ASSUM_TAC
              ("!t. tp' <= t /\ t < ti' ==> SND(I_cgnt_E(e' t))","tp':timeC")
       THEN RES_TAC
       THEN ASM_REWRITE_ASSUM_TAC ("SND(I_cgnt_E(e' (tp':timeC)))",[])
       THEN NRULE_ASSUM_TAC
              ("STABLE_FALSE_THEN_TRUE(ale_sig_ib p')(tp',u)",
               (BETA_RULE o (REWRITE_RULE [STABLE_FALSE_THEN_TRUE])))
        THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
        THEN UNDISCH_TAC "ale_sig_ib p' u"
        THEN REWRITE_TAC [ale_sig_ib;BSel]
        THEN BETA_TAC
        THEN IMP_RES_TAC I_male_ISO
        THEN IMP_RES_TAC I_rale_ISO
        THEN IMP_RES_TAC I_cale_ISO
        THEN ASM_REWRITE_TAC[WIRE]
        THEN COND_CASES_TAC
        THEN ASM_REWRITE_TAC [prove_constructors_distinct wire;
                              SYM_RULE (prove_constructors_distinct wire)]
     ;
        % Subgoal 2: (Continue) - [ "tp' < u" ] - %
        IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LT_IMP_LE_PRE)
        THEN IMP_RES_TAC STABLE_FALSE_THEN
        THEN ASSUME_TAC (SPEC "u:timeC" LESS_EQ_REFL)
        THEN IMP_RES_TAC NEW_P_DEST1_STABLE_FROM_TP'_TO_TI'
        THEN NRULE_ASSUM_TAC
               ("STABLE_TRUE_THEN_FALSE(bsig I_cgnt_E e')(tp',ti')",
                (BETA_RULE o (REWRITE_RULE [STABLE_TRUE_THEN_FALSE;bsig;BSel])))
        THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
        THEN SPEC_ASSUM_TAC
               ("!t. tp' <= t /\ t < ti' ==> SND(I_cgnt_E(e' t))","u:timeC")
        THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
        THEN IMP_RES_TAC LESS_EQ_TRANS
        THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LE_PRE_IMP_LT)
        THEN RES_TAC
        THEN REWRITE_ASSUM_TAC
               ("STABLE_FALSE_THEN_TRUE(ale_sig_ib p')(tp',u)",
                [STABLE_FALSE_THEN_TRUE])
        THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
        THEN UNDISCH_TAC "ale_sig_ib p' u"
        THEN UNDISCH_TAC "P_dest1S(s'(u + 1))"
        THEN IMP_RES_TAC P_dest1_ISO
        THEN ASM_REWRITE_TAC[]
        THEN DISCH_TAC
        THEN REWRITE_TAC [ale_sig_ib;BSel]
        THEN BETA_TAC
        THEN IMP_RES_TAC I_male_ISO
        THEN IMP_RES_TAC I_rale_ISO
        THEN IMP_RES_TAC I_cale_ISO
        THEN ASM_REWRITE_TAC [WIRE]
        THEN COND_CASES_TAC
        THEN ASM_REWRITE_TAC [prove_constructors_distinct wire;
                              SYM_RULE (prove_constructors_distinct wire)]
     ]
    );;

let ALE_SIG_IB_TRUE_AFTER_TP' = TAC_PROOF
   (([],
    "! (t :timeT) (pti :PTI)
       (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' :timeC) .
   PCSet_Correct s' e' p' ==>
     NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
       (tp' > 0) ==>
         PT_Exec pti s e p t ==>
           PT_PreC pti s e p t ==>
             PStateAbs pti s e p t s' e' p' tp' ==>
               IBA_PMaster pti e p t e' p' ==>
                 Rst_Slave pti e t e' ==>
                   ~(~ELEMENT (FST(L_ad_inE (e' tp'))) 31 /\
                     New_State_Is_PA s' e' tp') ==>
                       (?ti':timeC. STABLE_FALSE_THEN_TRUE
                                     (ale_sig_ib p') (tp',ti'))"),
    REWRITE_TAC [DE_MORGAN_THM]
    THEN PURE_ONCE_REWRITE_TAC [DISJOINT_OR]
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC IBA_READY_ASSUMPS
    THEN IMP_RES_TAC PREC
    THEN RES_TAC
```

```
THENL [
   % Subgoal 1: [ "ELEMENT(FST(L_ad_inE(e' tp')))31" ] %
   SUBGOAL_THEN
      "CHANGES_FALSE (bsig I_crqt_O p') tp'"
      ASSUME_TAC
   THENL [
      % Subgoal 1.1: "CHANGES_FALSE (bsig I_crqt_O p')tp'" %
      REWRITE_TAC [CHANGES_FALSE;bsig;BSel]
      THEN BETA_TAC
      THEN CONJ_TAC
      THENL [
         % Subgoal 1.1.1: "(tp' = 0) \/ SND(I_crqt_O(p'(tp' - 1)))" %
         DISJ2_TAC
         THEN PREC_TAC
         THEN REWRITE_ASSUM_TAC
                ("tp' > 0",[SYM_RULE ONE_LESS_EQ])
         THEN IMP_RES_TAC (SPECL ["tp':timeC";"1"] SUB_ADD)
         THEN IMP_RES_TAC (SYM_RULE P_rqt_ISO)
         THEN SPEC_UNDISCH_MATCH_RHS_TAC
                ("P_rqtS(s'(t + 1))","tp'-1")
         THEN ASM_REWRITE_TAC[]
         THEN DISCH_TAC
         THEN IMP_RES_TAC I_crqt_ISO
         THEN ASM_REWRITE_TAC[]
      ;
         % Subgoal 1.1.2: "~SND(I_crqt_O(p' tp'))" %
         PREC_TAC
         THEN IMP_RES_TAC RST_FALSE
         THEN SPEC_ASSUM_TAC ("!u':timeC. SND(RstE(e' u')) = F","tp':timeC")
         THEN IMP_RES_TAC P_RQT_PREVENTS_NEW_STATE_PD
         THEN IMP_RES_TAC NTH_ALE_SIG_PB_TRUE
         THEN REWRITE_ASSUM_TAC
                ("~New_State_Is_PD s' e' tp'",[New_State_Is_PD])
         THEN IMP_RES_TAC I_crqt_ISO
         THEN ASM_REWRITE_TAC[]
      ]
   ;
      % Subgoal 1.2:  [ "CHANGES_FALSE(bsig I_crqt_O p')tp'" ] %
      RES_TAC
      THEN IMP_RES_TAC ALE_SIG_IB_FALSE_AWAITING_CGNT
      THEN NRULE_ASSUM_TAC
             ("STABLE_TRUE_THEN_FALSE(bsig I_cgnt_E e')(tp',v')",
              (BETA_RULE o
               (REWRITE_RULE [STABLE_TRUE_THEN_FALSE;bsig;BSel])))
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN SUBGOAL_THEN
             "~SND(I_cgnt_E(e' v')) ==>
                (SND(I_hold_E(e' v')) /\ New_State_Is_PA s' e' v')"
             IMP_RES_TAC
      THENL [
         % Subgoal 1.2.1:
             "~SND(I_cgnt_E(e' v')) ==>
               SND(I_hold_E(e' v')) /\ New_State_Is_PA s' e' v'" %
         SPEC_ASSUM_TAC
           ("!p'. STABLE_FALSE(ale_sig_ib p')(tp',v' - 1)",
            "p':timeC->pc_out")
         THEN ASSUME_TAC (SPEC "v':timeC" LESS_EQ_REFL)
         THEN IMP_RES_TAC SUB_LESS_OR
         THEN ASSUME_TAC (SPECL ["v':timeC";"1"] SUB_LESS_EQ)
         THEN IMP_RES_TAC NEW_P_DEST1_STABLE_FROM_TP'_TO_TI'
         THEN IMP_RES_TAC NEW_STATE_PD_FALSE_FROM_TP'_TO_TI'
         THEN IMP_RES_TAC NEW_P_RQT_TRUE_FROM_TP'_TO_TI'
         THEN DISCH_TAC
         THEN POP_ASSUM_LIST
                (MAP_EVERY (\thm. ASSUME_TAC (REWRITE_RULE [BSel] thm)))
         THEN RES_TAC
         THEN ASM_REWRITE_TAC[New_State_Is_PA]
         THEN COND_CASES_TAC
         THEN ASM_REWRITE_TAC[]
         THEN REWRITE_ASSUM_TAC ("tp' > 0",[SYM_RULE ONE_LESS_EQ])
         THEN IMP_RES_TAC LESS_EQ_TRANS
         THEN IMP_RES_TAC (SPECL ["v':timeC";"1"] SUB_ADD)
```

```
            THEN IMP_RES_TAC P_fsm_hold_ISO
            THEN ASM_REWRITE_ASSUM_TAC ("P_fsm_hold_S(s'((v' - 1) + 1))",[])
            THEN ASM_REWRITE_TAC[]
            THEN IMP_RES_TAC
                   (SPECL ["s':timeC->pc_state";"e':timeC->pc_env";"v'-1"]
                          NEXT_STATE_NOT_PD)
            THENL [
               % Subgoal 1.2.1.1:  [ "New_State_Is_PA s' e'(v' - 1)" ] %
               UNDISCH_TAC "New_State_Is_PA s' e'(v' - 1)"
               THEN IMP_RES_TAC P_fsm_state_ISO
               THEN SPEC_UNDISCH_MATCH_LHS_TAC
                      ("P_fsm_stateS(s'(t + 1))","v'-1")
               THEN ASM_REWRITE_TAC [New_State_Is_PA]
               THEN DISCH_TAC
               THEN DISCH_TAC
               THEN ASM_REWRITE_TAC
                      [SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
               THEN POP_ASSUM (\thm. ALL_TAC)
               THEN POP_ASSUM (\thm. ALL_TAC)
               THEN SPEC_ASSUM_TAC
                      ("!t. tp' <= t /\ t < v' ==> SND(I_cgnt_E(e' t))","v'-1")
               THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LE_IMP_PRE_LT)
               THEN RES_TAC
               THEN IMP_RES_TAC P_fsm_cgnt_ISO
               THEN UNDISCH_TAC "P_fsm_cgnt_S(s'((v' - 1) + 1))"
               THEN FILTER_ASM_REWRITE_TAC
                      (\tm. not (tm = "(v' - 1) + 1 = v'")) []
               THEN ASM_REWRITE_TAC[]
               THEN DISCH_TAC
               THEN ASM_REWRITE_TAC[]
               THEN ASM_REWRITE_ASSUM_TAC
                      ("P_dest1S(s'((v' - 1) + 1))",[])
               THEN IMP_RES_TAC (LIMP ONE_LESS_EQ)
               THEN IMP_RES_TAC P_DEST1_TRUE_IMP_P_FSM_MRQT_FALSE
               THEN ASM_REWRITE_TAC[]
            ;
               % Subgoal 1.2.1.2:  [ "New_State_Is_PH s' e'(v' - 1)" ] %
               UNDISCH_TAC "New_State_Is_PH s' e'(v' - 1)"
               THEN IMP_RES_TAC P_fsm_state_ISO
               THEN SPEC_UNDISCH_MATCH_LHS_TAC
                    ("P_fsm_stateS(s'(t + 1))","v'-1")
               THEN ASM_REWRITE_TAC [New_State_Is_PH]
               THEN DISCH_TAC
               THEN DISCH_TAC
               THEN ASM_REWRITE_TAC
                      [SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
            ]
         ;
            % Subgoal 1.2.2:  [ "SND(I_hold_E(e' v'))" ]
                              [ "New_State_Is_PA s' e' v'" ] %
            EXISTS_TAC "v':timeC"
            THEN REWRITE_TAC [STABLE_FALSE_THEN_TRUE;ale_sig_ib;BSel]
            THEN BETA_TAC
            THEN ASM_REWRITE_TAC[]
            THEN CONJ_TAC
            THENL [
               % Subgoal 1.2.2.1: "!t. tp' <= t /\ t < v' ==>
                                  ~(SND(I_hlda_O(p' t)) /\
                                    ((SND(I_male_O(p' t)) = LO) \/
                                     (SND(I_rale_O(p' t)) = LO) \/
                                    ~SND(I_cale_O(p' t))))" %
%
               NRULE_ASSUM_TAC
                 ("!p'. STABLE_FALSE(ale_sig_ib p')(tp',v' - 1)",
                  ((SPEC "p':timeC->pc_out") o BETA_RULE o
                   (REWRITE_RULE [STABLE_FALSE;ale_sig_ib;BSel])))
%
               NRULE_ASSUM_TAC
                 ("STABLE_FALSE(ale_sig_ib p')(tp',v' - 1)",
                  (BETA_RULE o (REWRITE_RULE [STABLE_FALSE;ale_sig_ib;BSel])))
               THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
               THEN GEN_TAC
            THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
            THEN IMP_RES_TAC LESS_EQ_TRANS
            THEN ASSUME_TAC
              (REWRITE_RULE [PRE_SUB1]
                            (SPECL ["t':timeC";"v':timeC"] LT_EQ_LE_PRE))
            THEN RES_TAC
            THEN ASM_REWRITE_TAC[]
          ;
            % Subgoal 1.2.2.2: " ~SND(I_hlda_O(p' v')) /\
                                ((SND(I_male_O(p' v')) = LO) \/
                                (SND(I_rale_O(p' v')) = LO) \/
                                ~SND(I_cale_O(p' v')))" %
            IMP_RES_TAC I_hlda_ISO
            THEN IMP_RES_TAC I_cale_ISO
            THEN REWRITE_ASSUM_TAC
                   ("New_State_Is_PA s' e' v'",[New_State_Is_PA])
            THEN ASM_REWRITE_TAC
                   [SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
          ]
        ]
      ]
    ;
      % Subgoal 2: [ "~ELEMENT(FST(L_ad_inE(e' tp')))31" ]
                   [ "~New_State_Is_PA s' e' tp'" ] %
      IMP_RES_TAC NEXT_STATE_NOT_PA
      THENL [
         % Subgoal 2.1: [ "new state = PD" ] %
         REWRITE_ASSUM_TAC
           ("PStateAbs pti s e p t s' e' p' tp'",[PStateAbs])
         THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
         THEN RES_TAC
         THEN REWRITE_ASSUM_TAC ("New_State_Is_PD s' e' tp'",[New_State_Is_PD])
         THEN ASM_REWRITE_ASSUM_TAC
                ("P_rqtS(s' (tp':timeC)) = PT_rqtS(s (t:timeT))",[])
         THEN IMP_RES_TAC (SPEC "PT_fsm_stateS(s (t:timeT))" NOT_PD)
         THEN ASM_REWRITE_ASSUM_TAC
                ("P_fsm_stateS(s' (tp':timeC)) = PT_fsm_stateS(s (t:timeT))",[])
         THEN IMP_RES_TAC PA_IMP_NOT_PD
         THEN IMP_RES_TAC PH_IMP_NOT_PD
         THEN IMP_RES_TAC
                (REWRITE_RULE [New_State_Is_PD] P_RQT_PREVENTS_NEW_STATE_PD)
      ;
         % Subgoal 2.2: [ "new state = PH" ] %
         IMP_RES_TAC EVENTUALLY_PA_AFTER_PH
         THEN SUBGOAL_THEN
                "STABLE_FALSE_THEN_TRUE(\v'. New_State_Is_PA s' e' v')(tp',u')
                 ==> (tp'<u')
                 ==> STABLE_FALSE(ale_sig_ib p')(tp',(u'-1))"
                IMP_RES_TAC
         THENL [
            % Subgoal 2.2.1: (New subgoal) %
            REWRITE_TAC [STABLE_FALSE_THEN_TRUE;STABLE_TRUE;ale_sig_ib;BSel;
                         STABLE_FALSE]
            THEN BETA_TAC
            THEN REPEAT STRIP_TAC
            THEN IMP_RES_TAC SUB_LESS_OR
            THEN SPEC_ASSUM_TAC
                   ("!t. tp' <= t /\ t < u' ==> ~New_State_Is_PA s' e' t",
                    "t':timeC")
            THEN REWRITE_ASSUM_TAC ("tp' > 0",[SYM_RULE ONE_LESS_EQ])
            THEN IMP_RES_TAC LESS_EQ_TRANS
            THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LE_PRE_IMP_LT)
            THEN RES_TAC
            THENL [
               % Subgoal 2.2.1.1: [ "SND(I_male_O(p' t')) = LO" ] %
               UNDISCH_TAC "SND(I_male_O(p' (t':timeC))) = LO"
               THEN IMP_RES_TAC I_male_ISO
            ;
               % Subgoal 2.2.1.2:  [ "SND(I_rale_O(p' t')) = LO" ] %
               UNDISCH_TAC "SND(I_rale_O(p' (t':timeC))) = LO"
               THEN IMP_RES_TAC I_rale_ISO
            ;
      % Subgoal 2.2.1.3: [ "~SND(I_cale_O(p' t'))" ] %
      UNDISCH_TAC "~SND(I_cale_O(p' (t':timeC)))"
      THEN IMP_RES_TAC I_cale_ISO
   ]
   THEN REWRITE_ASSUM_TAC
          ("~New_State_Is_PA s' e' t'",[New_State_Is_PA])
   THEN ASM_REWRITE_TAC[WIRE]
   THEN COND_CASES_TAC
   THEN ASM_REWRITE_TAC
          [SYM_RULE (prove_constructors_distinct wire);
           prove_constructors_distinct wire]
;
   % Subgoal 2.2.2 %
   IMP_RES_TAC NEW_P_DEST1_STABLE_FROM_TP'_TO_TI'
   THEN POP_ASSUM (\thm. ALL_TAC)
   THEN POP_ASSUM (\thm. ALL_TAC)
   THEN IMP_RES_TAC NEW_P_RQT_TRUE_FROM_TP'_TO_TI'
   THEN SPEC_ASSUM_TAC
          ("!t'. tp' <= t' ==>
                   t' <= u' ==>
                     (P_dest1S(s'(t' + 1)) =
                       ELEMENT(FST(L_ad_inE(e' tp')))31)","u'-1")
   THEN SPEC_ASSUM_TAC
          ("!t'. tp' <= t' ==>
                   t' <= u' ==>
                     New_P_Rqt_Is_TRUE s' e' t'","u'-1")
   THEN ASSUME_TAC (SPEC "u':timeC" LESS_EQ_REFL)
   THEN ASSUME_TAC (SPEC "u'-1" LESS_EQ_REFL)
   THEN NRULE_ASSUM_TAC
     ("STABLE_FALSE(ale_sig_ib p')(tp',u' - 1)",
      (BETA_RULE o (REWRITE_RULE [STABLE_FALSE])))
   THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
   THEN ASSUME_TAC (SPECL ["u':timeC";"1"] SUB_LESS_EQ)
   THEN REWRITE_ASSUM_TAC ("tp' > 0",[SYM_RULE ONE_LESS_EQ])
   THEN IMP_RES_TAC LESS_EQ_TRANS
   THEN RES_TAC
   THEN IMP_RES_TAC (SPECL ["u':timeC";"1"] SUB_ADD)
   THEN ASM_REWRITE_ASSUM_TAC
          ("P_dest1S(s'((u' - 1) + 1)) =
              ELEMENT(FST(L_ad_inE(e' (tp':timeC))))31",[])
   THEN NRULE_ASSUM_TAC
         ("STABLE_FALSE_THEN_TRUE(\v'. New_State_Is_PA s' e' v')
           (tp',u')",
          (BETA_RULE o (REWRITE_RULE [STABLE_FALSE_THEN_TRUE])))
   THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
   THEN EXISTS_TAC "u':timeC"
   THEN REWRITE_TAC [STABLE_FALSE_THEN_TRUE;ale_sig_ib;BSel]
   THEN BETA_TAC
   THEN ASM_REWRITE_TAC[]
   THEN CONJ_TAC
   THENL [
      % Subgoal 2.2.2.1: "!t. tp' <= t /\ t < u' ==>
                                 ~(SND(I_hlda_O(p' t)) /\
                                   ((SND(I_male_O(p' t)) = LO) \/
                                    (SND(I_rale_O(p' t)) = LO) \/
                                    ~SND(I_cale_O(p' t))))" %
      GEN_TAC
      THEN ASSUME_TAC
        (REWRITE_RULE [PRE_SUB1]
                      (SPECL ["t':timeC";"u':timeC"] LT_EQ_LE_PRE))
      THEN RES_TAC
      THEN ASM_REWRITE_TAC[]
      THEN NRULE_ASSUM_TAC
             ("!t. tp' <= t /\ t <= (u' - 1) ==> ~ale_sig_ib p' t",
              (BETA_RULE o (REWRITE_RULE [ale_sig_ib;BSel])))
      THEN ASM_REWRITE_TAC[]
   ;
      % Subgoal 2.2.2.2: "SND(I_hlda_O(p' u')) /\
                            ((SND(I_male_O(p' u')) = LO) \/
                             (SND(I_rale_O(p' u')) = LO) \/
                             ~SND(I_cale_O(p' u')))" %
      CONJ_TAC
```

```
                  THENL [
                     % Subgoal 2.2.2.2.1: "SND(I_hlda_O(p' u'))" %
                     IMP_RES_TAC I_hlda_ISO
                     THEN REWRITE_ASSUM_TAC
                            ("New_State_Is_PA s' e' u'",[New_State_Is_PA])
                     THEN ASM_REWRITE_TAC
                            [SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
                  ;
                     % Subgoal 2.2.2.2.2: "(SND(I_male_O(p' u')) = LO) \/
                                           (SND(I_rale_O(p' u')) = LO) \/
                                           ~SND(I_cale_O(p' u'))" %
                     IMP_RES_TAC RST_FALSE
                     THEN SPEC_ASSUM_TAC
                            ("!u':timeC. SND(RstE(e' u')) = F","u':timeC")
                     THEN IMP_RES_TAC P_rqt_ISO
                     THEN SPEC_UNDISCH_MATCH_LHS_TAC ("P_rqtS(s'(t + 1))","u'-1")
                     THEN ASM_REWRITE_TAC[]
                     THEN DISCH_TAC
                     THEN ASM_REWRITE_ASSUM_TAC
                            ("New_P_Rqt_Is_TRUE s' e' (u'-1)",
                             [New_P_Rqt_Is_TRUE;New_State_Is_PD])
                     THEN IMP_RES_TAC I_male_ISO
                     THEN IMP_RES_TAC I_rale_ISO
                     THEN REWRITE_ASSUM_TAC
                            ("New_State_Is_PA s' e' u'",[New_State_Is_PA])
                     THEN ASM_REWRITE_TAC
                            [SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom);
                             prove_constructors_distinct pfsm_ty_Axiom;WIRE;
                             COND_TRUE_TRUE;COND_TRUE_CHOICES]
                     THEN ASM_CASES_TAC
                            "SUBARRAY(P_addrS(s' (u':timeC)))(25,24) = WORDN 1 3"
                     THEN ASM_REWRITE_TAC[]
                  ]
               ]
            ]
         ]
      ]
     );;

let TI'_AFTER_TP' = TAC_PROOF
   (([],
    "! (t :timeT) (pti :PTI)
       (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' :timeC) .
     PCSet_Correct s' e' p' ==>
      NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
       (tp' > 0) ==>
        PT_Exec pti s e p t ==>
         IBA_PMaster pti e p t e' p' ==>
          Rst_Slave pti e t e' ==>
           PT_PreC pti s e p t ==>
            PStateAbs pti s e p t s' e' p' tp' ==>
             STABLE_FALSE_THEN_TRUE (ale_sig_ib p') (tp',ti') ==>
              (ELEMENT (FST(L_ad_inE (e' tp'))) 31 \/
                         ~New_State_Is_PA s' e' tp') ==>
               (tp' < ti')"),
    REWRITE_TAC [STABLE_FALSE_THEN_TRUE;LESS_OR_EQ]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
    THENL [
       % Subgoal 1: [ "ELEMENT(FST(L_ad_inE(e' tp')))31" ] %
       IMP_RES_TAC IBA_READY_ASSUMPS
       THEN SUBGOAL_THEN
              "CHANGES_FALSE (bsig I_crqt_O p')tp'" ASSUME_TAC
       THENL [
          % Subgoal 1.1: "CHANGES_FALSE (bsig I_crqt_O p')tp'" %
          REWRITE_TAC [CHANGES_FALSE;bsig;BSel]
          THEN BETA_TAC
          THEN CONJ_TAC
          THENL [
             % Subgoal 1.1.1: "(tp' = 0) \/ SND(I_crqt_O(p'(tp' - 1)))" %
        DISJ2_TAC
        THEN PREC_TAC
        THEN REWRITE_ASSUM_TAC ("tp' > 0",[SYM_RULE ONE_LESS_EQ])
        THEN IMP_RES_TAC (SPECL ["tp':timeC";"1"] SUB_ADD)
        THEN IMP_RES_TAC (SYM_RULE P_rqt_ISO)
        THEN SPEC_UNDISCH_MATCH_RHS_TAC ("P_rqtS(s'(t + 1))","tp'-1")
        THEN ASM_REWRITE_TAC[]
        THEN DISCH_TAC
        THEN IMP_RES_TAC I_crqt_ISO
        THEN ASM_REWRITE_TAC[]
    ;
        % Subgoal 1.1.2: "~SND(I_crqt_O(p' tp'))" %
        PREC_TAC
        THEN IMP_RES_TAC RST_FALSE
        THEN SPEC_ASSUM_TAC ("!u':timeC. SND(RstE(e' u')) = F","ti':timeC")
        THEN IMP_RES_TAC P_RQT_PREVENTS_NEW_STATE_PD
        THEN UNDISCH_TAC
              "~P_rqtS(s' tp') ==> tp' > 0 ==> ~New_State_Is_PD s' e' tp'"
        THEN FILTER_ASM_REWRITE_TAC
                (\tm. not (tm = "P_fsm_stateS(s' (tp':timeC)) =
                                 PT_fsm_stateS(s (t:timeT))"))
                [New_State_Is_PD]
        THEN DISCH_TAC
        THEN IMP_RES_TAC NTH_ALE_SIG_PB_TRUE
        THEN ASM_REWRITE_ASSUM_TAC ("~SND(L_ads_E(e' (tp':timeC)))",[])
        THEN ASM_REWRITE_ASSUM_TAC ("SND(L_den_E(e' (tp':timeC)))",[])
        THEN ASM_REWRITE_ASSUM_TAC
              ("ELEMENT(FST(L_ad_inE(e' (tp':timeC))))31",[])
        THEN IMP_RES_TAC I_crqt_ISO
        THEN FILTER_ASM_REWRITE_TAC
                (\tm. not (tm = "P_fsm_stateS(s' (tp':timeC)) =
                                 PT_fsm_stateS(s (t:timeT))"))
                [New_State_Is_PD]
    ]
  ;
    % Subgoal 1.2: [ "CHANGES_FALSE(bsig I_crqt_O p')tp'" ] %
    RES_TAC
    THEN NRULE_ASSUM_TAC
          ("STABLE_TRUE_THEN_FALSE(bsig I_cgnt_E e')(tp',v')",
           (BETA_RULE o (REWRITE_RULE [STABLE_TRUE_THEN_FALSE;bsig;BSel])))
    THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
    THEN SPEC_ASSUM_TAC
          ("!t. tp' <= t /\ t < v' ==> SND(I_cgnt_E(e' t))","tp':timeC")
    THEN ASSUME_TAC (SPEC "tp':timeC" LESS_EQ_REFL)
    THEN RES_TAC
    THEN ASM_REWRITE_ASSUM_TAC ("SND(I_cgnt_E(e' (tp':timeC)))",[])
    THEN IMP_RES_TAC PREC
    THEN REWRITE_ASSUM_TAC
          ("PStateAbs pti s e p t s' e' p' tp'",[PStateAbs])
    THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
    THEN RES_TAC
    THEN ASM_REWRITE_ASSUM_TAC
          ("P_rqtS(s' (tp':timeC)) = PT_rqtS(s (t:timeT))",[])
    THEN ASM_REWRITE_ASSUM_TAC
          ("ELEMENT(FST(L_ad_inE(e' (tp':timeC))))31",[])
    THEN DELETE_ASSUM_TAC
          "P_fsm_stateS(s' (tp':timeC)) = PT_fsm_stateS(s (t:timeT))"
    THEN UNDISCH_TAC "ale_sig_ib p' ti'"
    THEN REWRITE_TAC [ale_sig_ib;BSel]
    THEN BETA_TAC
    THEN IMP_RES_TAC I_male_ISO
    THEN IMP_RES_TAC I_rale_ISO
    THEN IMP_RES_TAC I_cale_ISO
    THEN ASM_REWRITE_TAC []
    THEN ASM_CASES_TAC "New_State_Is_PH s' e' ti'"
    THEN REWRITE_ASSUM_TAC ("New_State_Is_PH s' e' ti'",[New_State_Is_PH])
    THEN ASM_REWRITE_TAC [WIRE;prove_constructors_distinct wire;
                          SYM_RULE (prove_constructors_distinct wire)]
    THEN REWRITE_ASSUM_TAC ("~New_State_Is_PH s' e' ti'",[New_State_Is_PH])
    THEN ASM_REWRITE_TAC [WIRE;prove_constructors_distinct wire;
                          SYM_RULE (prove_constructors_distinct wire)]
]
```

```
         ;
            % Subgoal 2: [ "~New_State_Is_PA s' e' tp'" ] %
            IMP_RES_TAC IBUS_ALE_IMP_PA
            THEN ASM_REWRITE_ASSUM_TAC ("~New_State_Is_PA s' e' tp'",[])
            THEN RES_TAC
         ]
        );;

    let IBUS_TRANS_EXISTS = TAC_PROOF
       (([],
        "! (t :timeT) (pti :PTI)
            (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
            (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
            (tp' :timeC) .
       PCSet_Correct s' e' p' ==>
         NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
            (tp' > 0) ==>
              PT_Exec pti s e p t ==>
                PT_PreC pti s e p t ==>
                  PStateAbs pti s e p t s' e' p' tp' ==>
                    IBA_PMaster pti e p t e' p' ==>
                      Rst_Slave pti e t e' ==>
                        (?ti':timeC. STABLE_FALSE_THEN_TRUE
                                       (ale_sig_ib p') (tp',ti'))"),
        REPEAT STRIP_TAC
        THEN ASM_CASES_TAC
                "~ELEMENT (FST(L_ad_inE (e' tp'))) 31 /\ New_State_Is_PA s' e' tp'"
        THENL [
            % Subgoal 1: [ "~ELEMENT(FST(L_ad_inE(e' tp')))31 /\
                             New_State_Is_PA s' e' tp'" ] %
            EXISTS_TAC "tp':timeC"
            THEN REWRITE_TAC [STABLE_FALSE_THEN_TRUE]
            THEN BETA_TAC
            THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
            THEN IMP_RES_TAC ALE_SIG_IB_TRUE_ON_TP'
            THEN ASSUME_TAC (SPEC "tp':timeC" LESS_EQ_REFL)
            THEN ASM_REWRITE_TAC[]
            THEN GEN_TAC
            THEN ASM_CASES_TAC "tp' <= t'"
            THENL [
               % Subgoal 1.1: [ "tp' <= t'" ] %
               IMP_RES_TAC (RIMP NOT_LESS)
               THEN ASM_REWRITE_TAC[]
            ;
               % Subgoal 1.2: [ "~tp' <= t'" ] %
               IMP_RES_TAC NOT_LESS_EQ_LESS
               THEN ASM_REWRITE_TAC[]
            ]
         ;
            % Subgoal 2: [ "~(~ELEMENT(FST(L_ad_inE(e' tp')))31 /\
                               New_State_Is_PA s' e' tp')" ] %
            IMP_RES_TAC ALE_SIG_IB_TRUE_AFTER_TP'
            THEN EXISTS_TAC "ti':timeC"
            THEN ASM_REWRITE_TAC[]
         ]
        );;

    let OFFSET_NEW_STATE_PD_FROM_TI'_TO_T'SACK = TAC_PROOF
       (([],
        "! (u' ti' t'sack :timeC) (s' :timeC->pc_state) (e' :timeC->pc_env)
            (p' :timeC->pc_out) .
         PCSet_Correct s' e' p' ==>
           (tp' > 0) ==>
             PT_Exec pti s e p t ==>
              PT_PreC pti s e p t ==>
               PStateAbs pti s e p t s' e' p' tp' ==>
                IBA_PMaster pti e p t e' p' ==>
                 Rst_Slave pti e t e' ==>
                  ale_sig_ib p' ti' ==>
                   STABLE_FALSE_THEN_TRUE (Sack_Sig_Is_TRUE s' e') (ti',t'sack) ==>
                    ((ti'+u'+1) <= t'sack) ==>
                     New_State_Is_PD s' e' (ti'+u'+1)"),
INDUCT_TAC
THENL [
   % Subgoal 1: (Base Case) %
   REWRITE_TAC [STABLE_FALSE_THEN_TRUE;Sack_Sig_Is_TRUE;ADD_CLAUSES]
   THEN BETA_TAC
   THEN REPEAT STRIP_TAC
   THEN IMP_RES_TAC IBUS_ALE_IMP_PA
   THEN SUBGOAL_THEN
          "New_State_Is_PA s' e' ti' ==> (P_fsm_stateS (s' (ti'+1)) = PA)"
          IMP_RES_TAC
   THENL [
      % Subgoal 1.1: (New subgoal) %
      REWRITE_TAC [New_State_Is_PA]
      THEN DISCH_TAC
      THEN IMP_RES_TAC P_fsm_state_ISO
      THEN SPEC_UNDISCH_MATCH_LHS_TAC ("P_fsm_stateS (s (t+1))","ti':timeC")
      THEN ASM_REWRITE_TAC []
      THEN DISCH_TAC
      THEN ASM_REWRITE_TAC[]
   ;  % Subgoal 1.2: [ "P_fsm_stateS(s'(ti' + 1)) = PA" ] %
      ASSUME_TAC
        (PURE_ONCE_REWRITE_RULE
           [ADD_SYM] (SPECL ["1";"ti':timeC"] LESS_EQ_ADD))
      THEN IMP_RES_TAC P_FSM_RST_FALSE
      THEN IMP_RES_TAC IBUS_ALE_IMP_FSM_RQT
      THEN ASM_REWRITE_TAC
             [New_State_Is_PD;
              SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
   ]
  ;
   % Subgoal 2: (Induction step) %
   REWRITE_TAC [ADD1;ADD_ASSOC]
   THEN POP_ASSUM_LIST
          (MAP_EVERY (\thm. ASSUME_TAC (REWRITE_RULE [ADD1;ADD_ASSOC] thm)))
   THEN REPEAT STRIP_TAC
   THEN ASSUME_TAC (SPECL ["(ti'+u')+1";"1"] LESS_EQ_ADD)
   THEN IMP_RES_TAC LESS_EQ_TRANS
   THEN RES_TAC
   THEN SUBGOAL_THEN
          "New_State_Is_PD s' e' ((ti'+u')+1) ==>
             (P_fsm_stateS (s' (((ti'+u')+1)+1)) = PD)"
          IMP_RES_TAC
   THENL [
      % Subgoal 2.1: (New subgoal) %
      REWRITE_TAC [New_State_Is_PD]
      THEN DISCH_TAC
      THEN IMP_RES_TAC P_fsm_state_ISO
      THEN SPEC_UNDISCH_MATCH_LHS_TAC ("P_fsm_stateS (s (t+1))","(ti'+u')+1")
      THEN ASM_REWRITE_TAC []
      THEN DISCH_TAC
      THEN ASM_REWRITE_TAC[]
   ;
      % Subgoal 2.2: [ "P_fsm_stateS(s'(((ti' + u') + 1) + 1)) = PD" ] %
      SUBGOAL_THEN "~P_fsm_sackS (s' (((ti'+u')+1)+1))" ASSUME_TAC
      THENL [
         % Subgoal 2.2.1: (New subgoal) %
         NRULE_ASSUM_TAC
            ("STABLE_FALSE_THEN_TRUE(Sack_Sig_Is_TRUE s' e')(ti',t'sack)",
             (BETA_RULE o (REWRITE_RULE [STABLE_FALSE_THEN_TRUE])))
         THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
         THEN SPEC_ASSUM_TAC
                ("!t. ti' <= t /\ t < t'sack ==>
                        ~Sack_Sig_Is_TRUE s' e' t","(ti'+u')+1")
         THEN SUBGOAL_THEN "ti' <= ((ti' + u') + 1)" ASSUME_TAC
         THENL [
            % Subgoal 2.2.1.1: (New subgoal) - "ti' <= ((ti' + u') + 1)" %
            REWRITE_TAC [ASSOC_ADD_ADD1;LESS_EQ_ADD]
         ;
            % Subgoal 2.2.1.2: (Continue) %
            SUBGOAL_THEN "((ti' + u') + 1) < t'sack" ASSUME_TAC
            THENL [
              % Subgoal 2.2.1.2.1: (New subgoal): "((ti'+u') + 1) < t'sack" %
```

```
                  ASSUME_TAC
                    (SPECL ["1";"ti'+u'"]
                           (PURE_ONCE_REWRITE_RULE [ADD_SYM] LESS_EQ_ADD))
                  THEN IMP_RES_TAC LESS_EQ_TRANS
                  THEN IMP_RES_TAC (REWRITE_RULE [ADD1] SUC_LE_IMP_LT)
                ;
                  % Subgoal 2.2.1.2.2: (Continue) %
                  RES_TAC
                  THEN NRULE_ASSUM_TAC
                         ("~Sack_Sig_Is_TRUE s' e'((ti' + u') + 1)",
                          (BETA_RULE o
                            (REWRITE_RULE [Sack_Sig_Is_TRUE;New_State_Is_PD])))
                  THEN IMP_RES_TAC P_fsm_sack_ISO
                  THEN ASM_REWRITE_TAC[]
                ]
              ]
            ;
              % Subgoal 2.2.2:  [ "~P_fsm_sackS(s'(((ti' + u') + 1) + 1))" ] %
              ASSUME_TAC
                (PURE_ONCE_REWRITE_RULE
                   [ADD_SYM] (SPECL ["1";"(ti'+u')+1"] LESS_EQ_ADD))
              THEN IMP_RES_TAC P_FSM_RST_FALSE
              THEN ASM_REWRITE_TAC
                     [New_State_Is_PD;prove_constructors_distinct pfsm_ty_Axiom;
                      SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
            ]
          ]
        ]
      );;

let NEW_STATE_PD_FROM_TI'_TO_T'SACK = TAC_PROOF
    (([],
     "! (t' ti' t'sack :timeC) (s' :timeC->pc_state) (e' :timeC->pc_env)
        (p' :timeC->pc_out) .
      PCSet_Correct s' e' p' ==>
        (tp' > 0) ==>
          PT_Exec pti s e p t ==>
           PT_PreC pti s e p t ==>
            PStateAbs pti s e p t s' e' p' tp' ==>
             IBA_PMaster pti e p t e' p' ==>
              Rst_Slave pti e t e' ==>
               ale_sig_ib p' ti' ==>
                STABLE_FALSE_THEN_TRUE (Sack_Sig_Is_TRUE s' e') (ti',t'sack) ==>
                 ((ti'+1) <= t') ==>
                  (t' <= t'sack) ==>
                   New_State_Is_PD s' e' t'"),
     REPEAT STRIP_TAC
     THEN IMP_RES_TAC (SPEC "t'-ti'+1" OFFSET_NEW_STATE_PD_FROM_TI'_TO_T'SACK)
     THEN SPECL_ASSUM_TAC
            ("!t' ti'''. (ti' + ((t' - (ti''' + 1)) + 1)) <= t'sack ==>
                           New_State_Is_PD s' e'(ti' + ((t' - (ti''' + 1)) + 1))",
             ["t':timeC";"ti':timeC"])
     THEN SUBGOAL_THEN
            "(ti' + ((t' - (ti' + 1)) + 1)) = t'"
            (\thm. RULE_ASSUM_TAC (REWRITE_RULE [thm]))
     THENL [
        % Subgoal 1: -New subgoal- "ti' + ((t' - (ti' + 1)) + 1) = t'" %
        REWRITE_TAC [SYM_RULE (ASSOC_SUB_SUB1)]
        THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "1<=1"))
        THEN IMP_RES_TAC SUC_LE_IMP_LE
        THEN SUBGOAL_THEN "1 <= (t' - ti')" ASSUME_TAC
        THENL [
           % Subgoal 1.1: "1 <= (t' - ti')"
                             [ "(ti' + 1) <= t'" ] %
           REWRITE_TAC
             [SYM_RULE (SPECL ["1";"t'-ti'";"ti':timeC"] LESS_EQ_MONO_ADD_EQ)]
           THEN ASSUME_TAC (SPEC "ti':timeC" LESS_EQ_REFL)
           THEN IMP_RES_TAC
                  (SPECL ["t':timeC";"ti':timeC";"ti':timeC"] ASSOC_SUB_ADD1)
           THEN ASM_REWRITE_TAC [SUB_EQUAL_0;ADD_CLAUSES]
           THEN PURE_ONCE_REWRITE_TAC [ADD_SYM]
           THEN ASM_REWRITE_TAC[]
        ;
           % Subgoal 1.2: [ "1 <= (t' - ti')" ] %
           IMP_RES_TAC (SPECL ["t'-ti'";"1";"1"] ASSOC_SUB_ADD1)
           THEN ASM_REWRITE_TAC[SUB_EQUAL_0;ADD_CLAUSES]
           THEN PURE_ONCE_REWRITE_TAC [ADD_SYM]
           THEN ASSUME_TAC (SPEC "ti':timeC" LESS_EQ_REFL)
           THEN IMP_RES_TAC
                  (SPECL ["t':timeC";"ti':timeC";"ti':timeC"] ASSOC_SUB_ADD1)
           THEN ASM_REWRITE_TAC [SUB_EQUAL_0;ADD_CLAUSES]
        ]
     ;
        % Subgoal 2 %
        RES_TAC
     ]
    );;

let OFFSET_NEW_STATE_PD_FROM_TI'_TO_T'SACK_1 = TAC_PROOF
    (([],
     "! (u' ti' t'sack :timeC) (s' :timeC->pc_state) (e' :timeC->pc_env)
        (p' :timeC->pc_out) .
      PCSet_Correct s' e' p' ==>
       (tp' > 0) ==>
         PT_Exec pti s e p t ==>
          PT_PreC pti s e p t ==>
           PStateAbs pti s e p t s' e' p' tp' ==>
            IBA_PMaster pti e p t e' p' ==>
             Rst_Slave pti e t e' ==>
              ale_sig_ib p' ti' ==>
               STABLE_FALSE (Sack_Sig_Is_TRUE s' e') (ti',t'sack-1) ==>
                ((ti'+u'+1) <= t'sack) ==>
                 New_State_Is_PD s' e' (ti'+u'+1)"),
     INDUCT_TAC
     THENL [
        % Subgoal 1: (Base Case) %
        REWRITE_TAC [STABLE_FALSE;Sack_Sig_Is_TRUE;ADD_CLAUSES]
        THEN BETA_TAC
        THEN REPEAT STRIP_TAC
        THEN IMP_RES_TAC IBUS_ALE_IMP_PA
        THEN SUBGOAL_THEN
               "New_State_Is_PA s' e' ti' ==> (P_fsm_stateS (s' (ti'+1)) = PA)"
               IMP_RES_TAC
        THENL [
           % Subgoal 1.1: (New subgoal) %
           REWRITE_TAC [New_State_Is_PA]
           THEN DISCH_TAC
           THEN IMP_RES_TAC P_fsm_state_ISO
           THEN SPEC_UNDISCH_MATCH_LHS_TAC ("P_fsm_stateS (s (t+1))","ti':timeC")
           THEN ASM_REWRITE_TAC []
           THEN DISCH_TAC
           THEN ASM_REWRITE_TAC[]
        ;  % Subgoal 1.2: [ "P_fsm_stateS(s'(ti' + 1)) = PA" ] %
           ASSUME_TAC
             (PURE_ONCE_REWRITE_RULE
                [ADD_SYM] (SPECL ["1";"ti':timeC"] LESS_EQ_ADD))
           THEN IMP_RES_TAC P_FSM_RST_FALSE
           THEN IMP_RES_TAC IBUS_ALE_IMP_FSM_RQT
           THEN ASM_REWRITE_TAC
                  [New_State_Is_PD;
                   SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
        ]
      ;
        % Subgoal 2: (Induction step) %
        REWRITE_TAC [ADD1;ADD_ASSOC]
        THEN POP_ASSUM_LIST
               (MAP_EVERY (\thm. ASSUME_TAC (REWRITE_RULE [ADD1;ADD_ASSOC] thm)))
        THEN REPEAT STRIP_TAC
        THEN ASSUME_TAC (SPECL ["(ti'+u')+1";"1"] LESS_EQ_ADD)
        THEN IMP_RES_TAC LESS_EQ_TRANS
        THEN RES_TAC
        THEN SUBGOAL_THEN
               "New_State_Is_PD s' e' ((ti'+u')+1) ==>
                  (P_fsm_stateS (s' (((ti'+u')+1)+1)) = PD)"
               IMP_RES_TAC
         THENL [
            % Subgoal 2.1: (New subgoal) %
            REWRITE_TAC [New_State_Is_PD]
            THEN DISCH_TAC
            THEN IMP_RES_TAC P_fsm_state_ISO
            THEN SPEC_UNDISCH_MATCH_LHS_TAC ("P_fsm_stateS (s (t+1))","(ti'+u')+1")
            THEN ASM_REWRITE_TAC []
            THEN DISCH_TAC
            THEN ASM_REWRITE_TAC[]
         ;
            % Subgoal 2.2: [ "P_fsm_stateS(s'(((ti' + u') + 1) + 1)) = PD" ] %
            SUBGOAL_THEN "~P_fsm_sackS (s' (((ti'+u')+1)+1))" ASSUME_TAC
            THENL [
               % Subgoal 2.2.1: (New subgoal) %
               IMP_RES_TAC
                 (SPECL ["((ti'+u')+1)+1";"t'sack:timeC";"1"] LESS_EQ_MONO_SUB)
               THEN ASSUME_TAC
                     (REWRITE_RULE [ADD_CLAUSES]
                       (REDUCE_RULE (SPECL ["(ti'+u')+1";"1";"1"] ASSOC_ADD_SUB1)))
               THEN ASM_REWRITE_ASSUM_TAC
                      ("((((ti' + u') + 1) + 1) - 1) <= (t'sack - 1)",[])
               THEN NRULE_ASSUM_TAC
                      ("STABLE_FALSE(Sack_Sig_Is_TRUE s' e')(ti',t'sack - 1)",
                       (BETA_RULE o (REWRITE_RULE [STABLE_FALSE])))
               THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
               THEN SPEC_ASSUM_TAC
                      ("!t. ti' <= t /\ t < (t'sack - 1) ==>
                              ~Sack_Sig_Is_TRUE s' e' t","(ti'+u')+1")
               THEN SUBGOAL_THEN "ti' <= ((ti' + u') + 1)" ASSUME_TAC
               THENL [
                  % Subgoal 2.2.1.1: (New subgoal) - "ti' <= ((ti' + u') + 1)" %
                  REWRITE_TAC [ASSOC_ADD_ADD1;LESS_EQ_ADD]
               ;
                  % Subgoal 2.2.1.2: (Continue) %
                  RES_TAC
                  THEN NRULE_ASSUM_TAC
                         ("~Sack_Sig_Is_TRUE s' e'((ti' + u') + 1)",
                          (BETA_RULE o
                             (REWRITE_RULE [Sack_Sig_Is_TRUE;New_State_Is_PD])))
                  THEN IMP_RES_TAC P_fsm_sack_ISO
                  THEN ASM_REWRITE_TAC[]
               ]
            ;
               % Subgoal 2.2.2:  [ "~P_fsm_sackS(s'(((ti' + u') + 1) + 1))" ] %
               ASSUME_TAC
                 (PURE_ONCE_REWRITE_RULE
                    [ADD_SYM] (SPECL ["1";"(ti'+u')+1"] LESS_EQ_ADD))
               THEN IMP_RES_TAC P_FSM_RST_FALSE
               THEN ASM_REWRITE_TAC
                      [New_State_Is_PD;prove_constructors_distinct pfsm_ty_Axiom;
                       SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
            ]
         ]
      ]
     );;

let NEW_STATE_PD_FROM_TI'_TO_T'SACK_1 = TAC_PROOF
   (([],
    "! (t' ti' t'sack :timeC) (s' :timeC->pc_state) (e' :timeC->pc_env)
       (p' :timeC->pc_out) .
     PCSet_Correct s' e' p' ==>
      (tp' > 0) ==>
        PT_Exec pti s e p t ==>
         PT_PreC pti s e p t ==>
          PStateAbs pti s e p t s' e' p' tp' ==>
           IBA_PMaster pti e p t e' p' ==>
            Rst_Slave pti e t e' ==>
             ale_sig_ib p' ti' ==>
              STABLE_FALSE (Sack_Sig_Is_TRUE s' e') (ti',t'sack-1) ==>
               ((ti'+1) <= t') ==>
                (t' <= t'sack) ==>
```

```
                 New_State_Is_PD s' e' t'"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC (SPEC "t'-ti'+1" OFFSET_NEW_STATE_PD_FROM_TI'_TO_T'SACK_1)
    THEN SPECL_ASSUM_TAC
           ("!t' ti'''. (ti' + ((t' - (ti''' + 1)) + 1)) <= t'sack ==>
                          New_State_Is_PD s' e'(ti' + ((t' - (ti''' + 1)) + 1))",
            ["t':timeC";"ti':timeC"])
    THEN SUBGOAL_THEN
           "(ti' + ((t' - (ti' + 1)) + 1)) = t'"
           (\thm. RULE_ASSUM_TAC (REWRITE_RULE [thm]))
    THENL [
       % Subgoal 1: -New subgoal- "ti' + ((t' - (ti' + 1)) + 1) = t'" %
       REWRITE_TAC [SYM_RULE (ASSOC_SUB_SUB1)]
       THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "1<=1"))
       THEN IMP_RES_TAC SUC_LE_IMP_LE
       THEN SUBGOAL_THEN "1 <= (t' - ti')" ASSUME_TAC
       THENL [
          % Subgoal 1.1: "1 <= (t' - ti')"
                          [ "(ti' + 1) <= t'" ] %
          REWRITE_TAC
            [SYM_RULE (SPECL ["1";"t'-ti'";"ti':timeC"] LESS_EQ_MONO_ADD_EQ)]
          THEN ASSUME_TAC (SPEC "ti':timeC" LESS_EQ_REFL)
          THEN IMP_RES_TAC
                 (SPECL ["t':timeC";"ti':timeC";"ti':timeC"] ASSOC_SUB_ADD1)
          THEN ASM_REWRITE_TAC [SUB_EQUAL_0;ADD_CLAUSES]
          THEN PURE_ONCE_REWRITE_TAC [ADD_SYM]
          THEN ASM_REWRITE_TAC[]
       ;
          % Subgoal 1.2: [ "1 <= (t' - ti')" ] %
          IMP_RES_TAC (SPECL ["t'-ti'";"1";"1"] ASSOC_SUB_ADD1)
          THEN ASM_REWRITE_TAC[SUB_EQUAL_0;ADD_CLAUSES]
          THEN PURE_ONCE_REWRITE_TAC [ADD_SYM]
          THEN ASSUME_TAC (SPEC "ti':timeC" LESS_EQ_REFL)
          THEN IMP_RES_TAC
                 (SPECL ["t':timeC";"ti':timeC";"ti':timeC"] ASSOC_SUB_ADD1)
          THEN ASM_REWRITE_TAC [SUB_EQUAL_0;ADD_CLAUSES]
       ]
    ;
       % Subgoal 2 %
       RES_TAC
    ]
   );;

let OFFSET_NEW_P_RQT_FALSE_FROM_T'SACK_TO_TP'SUC = TAC_PROOF
   (([],
    "! (u' t'sack tp'suc :timeC) (s' :timeC->pc_state) (e' :timeC->pc_env)
       (p' :timeC->pc_out) .
     PCSet_Correct s' e' p' ==>
      (tp' > 0) ==>
        PT_Exec pti s e p t ==>
         PT_PreC pti s e p t ==>
          PStateAbs pti s e p t s' e' p' tp' ==>
            IBA_PMaster pti e p t e' p' ==>
             Rst_Slave pti e t e' ==>
              Sack_Sig_Is_TRUE s' e' t'sack ==>
               STABLE_FALSE_THEN_TRUE (ale_sig_pb e') (t'sack,tp'suc) ==>
                ((t'sack+u') < tp'suc) ==>
                 ~New_P_Rqt_Is_TRUE s' e' (t'sack+u')"),
    INDUCT_TAC
    THENL [
       % Subgoal 1: (Base Case) %
       REWRITE_TAC
         [ADD_CLAUSES;STABLE_FALSE_THEN_TRUE;ale_sig_pb;BSel;Sack_Sig_Is_TRUE]
       THEN BETA_TAC
       THEN REPEAT STRIP_TAC
       THEN SPEC_ASSUM_TAC
              ("!t. t'sack <= t /\ t <= (tp'suc - 1) ==>
                    ~(~SND(L_ads_E(e' t)) /\ SND(L_den_E(e' t)))","t'sack:timeC")
       THEN ASSUME_TAC (SPEC "t'sack:timeC" LESS_EQ_REFL)
       THEN RES_TAC
       THEN UNDISCH_TAC "New_P_Rqt_Is_TRUE s' e' t'sack"
       THEN ASM_REWRITE_TAC [New_P_Rqt_Is_TRUE]
      ;
         % Subgoal 2: (Induction Step) %
         REWRITE_TAC [ADD1;ADD_ASSOC]
         THEN REPEAT STRIP_TAC
         THEN ASSUME_TAC (SPECL ["t'sack+u'";"1"] LESS_EQ_ADD)
         THEN IMP_RES_TAC LESS_EQ_LESS_TRANS
         THEN RES_TAC
         THEN SUBGOAL_THEN
                "~New_P_Rqt_Is_TRUE s' e' (t'sack + u') ==>
                     ~P_rqtS (s' ((t'sack + u') + 1))" IMP_RES_TAC
         THENL [
            % Subgoal 2.1: -New subgoal- "~New_P_Rqt_Is_TRUE s' e'(t'sack + u') ==>
                                           ~P_rqtS(s'((t'sack + u') + 1))" %
            REWRITE_TAC [New_P_Rqt_Is_TRUE;New_State_Is_PD]
            THEN BETA_TAC
            THEN DISCH_TAC
            THEN IMP_RES_TAC P_rqt_ISO
            THEN ASM_REWRITE_TAC []
         ;
            % Subgoal 2.2: [ "~P_rqtS(s'((t'sack + u') + 1))" ] %
            NRULE_ASSUM_TAC
               ("STABLE_FALSE_THEN_TRUE(ale_sig_pb e')(t'sack,tp'suc)",
                (BETA_RULE o (REWRITE_RULE [STABLE_FALSE_THEN_TRUE;ale_sig_pb;
                                            BSel])))
            THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
            THEN SPEC_ASSUM_TAC
                   ("!t. t'sack <= t /\ t <= (tp'suc - 1) ==>
                      ~(~SND(L_ads_E(e' t)) /\ SND(L_den_E(e' t)))","(t'sack+u')+1")
            THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LT_IMP_LE_PRE)
            THEN ASSUME_TAC (SPECL ["t'sack:timeC";"u':timeC"] LESS_EQ_ADD)
            THEN IMP_RES_TAC LESS_EQ_TRANS
            THEN RES_TAC
            THEN UNDISCH_TAC "New_P_Rqt_Is_TRUE s' e'((t'sack + u') + 1)"
            THEN ASM_REWRITE_TAC
                   [New_P_Rqt_Is_TRUE;COND_TRUE_TRUE;COND_FALSE_CHOICES]
         ]
      ]
     );;

let NEW_P_RQT_FALSE_FROM_T'SACK_TO_TP'SUC = TAC_PROOF
    (([],
      "! (t' t'sack tp'suc :timeC) (s' :timeC->pc_state) (e' :timeC->pc_env)
         (p' :timeC->pc_out) .
       PCSet_Correct s' e' p' ==>
         (tp' > 0) ==>
           PT_Exec pti s e p t ==>
            PT_PreC pti s e p t ==>
             PStateAbs pti s e p t s' e' p' tp' ==>
              IBA_PMaster pti e p t e' p' ==>
               Rst_Slave pti e t e' ==>
                Sack_Sig_Is_TRUE s' e' t'sack ==>
                 STABLE_FALSE_THEN_TRUE (ale_sig_pb e') (t'sack,tp'suc) ==>
                  (t'sack <= t') ==>
                   (t' < tp'suc) ==>
                    ~New_P_Rqt_Is_TRUE s' e' t'"),
      REPEAT STRIP_TAC
      THEN IMP_RES_TAC
             (SPEC "t'-t'sack" OFFSET_NEW_P_RQT_FALSE_FROM_T'SACK_TO_TP'SUC)
      THEN SPECL_ASSUM_TAC
             ("!t' t'sack''. (t'sack + (t' - t'sack'')) < tp'suc ==>
                             ~New_P_Rqt_Is_TRUE s' e'(t'sack + (t' - t'sack''))",
              ["t':timeC";"t'sack:timeC"])
      THEN IMP_RES_TAC
             (SPECL ["t':timeC";"t'sack:timeC"]
                    (PURE_ONCE_REWRITE_RULE [ADD_SYM] SUB_ADD))
      THEN ASM_REWRITE_ASSUM_TAC
             ("(t'sack + (t' - t'sack)) < tp'suc ==>
                  ~New_P_Rqt_Is_TRUE s' e'(t'sack + (t' - t'sack))",[])
     );;

let M_LESS_0_LESS = TAC_PROOF
    (([],
     "! m n . (m < n) ==> (0 < n)"),
     INDUCT_TAC
     THEN REPEAT STRIP_TAC
     THEN ASM_REWRITE_TAC[]
     THEN IMP_RES_TAC LT_IMP_LE
     THEN IMP_RES_TAC SUC_LE_IMP_LT
     THEN RES_TAC
   );;

let TRANS_ONTO = TAC_PROOF
   (([],
    "! (ti' tp'suc :timeC) (t :timeT)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out) .
     PCSet_Correct s' e' p' ==>
      (tp' > 0) ==>
       PT_Exec pti s e p t ==>
        PT_PreC pti s e p t ==>
         PStateAbs pti s e p t s' e' p' tp' ==>
          IBA_PMaster pti e p t e' p' ==>
           Rst_Slave pti e t e' ==>
            NTH_TIME_TRUE t (ale_sig_ib p') 0 ti' ==>
              STABLE_FALSE_THEN_TRUE (ale_sig_pb e') (ti'+1,tp'suc) ==>
                TRUE_THEN_STABLE_FALSE (ale_sig_ib p') (ti',tp'suc-1)"),
    REWRITE_TAC [TRUE_THEN_STABLE_FALSE]
    THEN BETA_TAC
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
    THENL [
       % Subgoal 1: "ti' <= (tp'suc - 1)" %
       UNDISCH_TAC "STABLE_FALSE_THEN_TRUE(ale_sig_pb e')(ti' + 1,tp'suc)"
       THEN REWRITE_TAC [STABLE_FALSE_THEN_TRUE]
       THEN BETA_TAC
       THEN REPEAT STRIP_TAC
       THEN IMP_RES_TAC (SPECL ["ti'+1";"tp'suc:timeC";"1"] LESS_EQ_MONO_SUB)
       THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "1<=1"))
       THEN IMP_RES_TAC
              (REWRITE_RULE [SUB_EQUAL_0;ADD_CLAUSES]
                           (SPECL ["ti':timeC";"1";"1"] ASSOC_ADD_SUB1))
       THEN ASM_REWRITE_ASSUM_TAC
              ("((ti' + 1) - 1) <= (tp'suc - 1)",[])
       THEN ASM_REWRITE_TAC[]
    ;
       % Subgoal 2: "ale_sig_ib p' ti'" %
       SUBGOAL_THEN
         "!t. NTH_TIME_TRUE t(ale_sig_ib p')0 ti' ==> ale_sig_ib p' ti'"
         IMP_RES_TAC
       THEN INDUCT_TAC
       THEN REWRITE_TAC [NTH_TIME_TRUE;STABLE_FALSE_THEN_TRUE;ZERO_LESS_EQ]
       THEN REPEAT STRIP_TAC
       THEN ASM_REWRITE_TAC[]
    ;
       % Subgoal 3: [ "ti' < t'" ]
                    [ "t' <= (tp'suc - 1)" ]
                    [ "ale_sig_ib p' t'" ] %
       ASM_CASES_TAC
          "?t'sack. Sack_Sig_Is_TRUE s' e' t'sack /\
                    (ti'<=t'sack) /\ (t'sack<=tp'suc-1)"
       THENL [
          % Subgoal 3.1: [ "?t'sack. Sack_Sig_Is_TRUE s' e' t'sack /\
                                     ti' <= t'sack /\
                                     t'sack <= tp'suc-1" ] %
          CHOOSE_ASSUM_TAC
            "?t'sack. Sack_Sig_Is_TRUE s' e' t'sack /\ ti' <= t'sack /\
                      t'sack <= (tp'suc - 1)"
          THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
          THEN SUBGOAL_THEN
                 "?u. ti' <= u /\ u <= (tp'suc-1) /\
                      STABLE_FALSE_THEN_TRUE (Sack_Sig_Is_TRUE s' e') (ti',u)"
                 ASSUME_TAC
          THENL [
            % Subgoal 3.1.1: (New Subgoal) %
```

```
IMP_RES_TAC FIRST_EXISTS
THEN EXISTS_TAC "u''':timeC"
THEN ASM_REWRITE_TAC[]
;
% Subgoal 3.1.2: (Continue) %
CHOOSE_ASSUM_TAC
  "?u. ti' <= u /\ u <= (tp'suc - 1) /\
       STABLE_FALSE_THEN_TRUE(Sack_Sig_Is_TRUE s' e')(ti',u)"
THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
THEN NRULE_ASSUM_TAC
       ("STABLE_FALSE_THEN_TRUE(Sack_Sig_Is_TRUE s' e')(ti',u')",
        (BETA_RULE o (REWRITE_RULE [STABLE_FALSE_THEN_TRUE])))
THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
THEN SUBGOAL_THEN
       "!t. NTH_TIME_TRUE t(ale_sig_ib p')0 ti' ==>
            ale_sig_ib p' ti'" IMP_RES_TAC
THENL [
   % Subgoal 3.1.2.1: (New subgoal) %
   INDUCT_TAC
   THEN REWRITE_TAC [NTH_TIME_TRUE;STABLE_FALSE_THEN_TRUE;
                     ZERO_LESS_EQ]
   THEN REPEAT STRIP_TAC
   THEN ASM_REWRITE_TAC[]
;
   % Subgoal 3.1.2.2: [ "ale_sig_ib p' ti'" ] %
   ASM_CASES_TAC "(ti'+1) <= u"
   THENL [
      % Subgoal 3.1.2.2.1: [ "(ti' + 1) <= u" ] %
      SUBGOAL_THEN
        "STABLE_FALSE_THEN_TRUE(ale_sig_pb e')(ti' + 1,tp'suc) ==>
         STABLE_FALSE_THEN_TRUE(ale_sig_pb e')(u,tp'suc)"
        IMP_RES_TAC
      THENL [
        % Subgoal 3.1.2.2.1.1: (New subgoal) %
        ASSUME_TAC (SPECL ["tp'suc:timeC";"1"] SUB_LESS_EQ)
        THEN IMP_RES_TAC LESS_EQ_TRANS
        THEN REWRITE_TAC [STABLE_FALSE_THEN_TRUE]
        THEN REPEAT STRIP_TAC
        THEN ASM_REWRITE_TAC[]
        THEN SPEC_ASSUM_TAC
            ("!t. (ti' + 1) <= t /\ t < tp'suc ==> ~ale_sig_pb e' t",
             "t'':timeC")
        THEN IMP_RES_TAC LESS_EQ_TRANS
        THEN RES_TAC
      ;
        % Subgoal 3.1.2.2.1.2: (Continue) %
        ASM_CASES_TAC "t' <= u"
          THENL [
          % Subgoal 3.1.2.2.1.2.1: [ "t' <= u" ] -New_State_Is_PD- %
          IMP_RES_TAC (REWRITE_RULE [ADD1] LT_IMP_SUC_LE)
          THEN IMP_RES_TAC NEW_STATE_PD_FROM_TI'_TO_T'SACK
          THEN IMP_RES_TAC IBUS_ALE_IMP_PA
          THEN REWRITE_ASSUM_TAC
                 ("New_State_Is_PA s' e' t'",[New_State_Is_PA])
          THEN UNDISCH_TAC "New_State_Is_PD s' e' t'"
          THEN ASM_REWRITE_TAC
                 [New_State_Is_PD;
                  prove_constructors_distinct pfsm_ty_Axiom]
        ;
          % Subgoal 3.1.2.2.1.2.2: [ "~t' <= u" ]
                                     -New_P_Rqt_Is_FALSE-%
          IMP_RES_TAC NOT_LESS_EQ_LESS
          THEN IMP_RES_TAC LT_IMP_LE
          THEN IMP_RES_TAC M_LESS_0_LESS
          THEN IMP_RES_TAC (RIMP(REWRITE_RULE [GREATER] ONE_LESS_EQ))
          THEN ASSUME_TAC (SPECL ["tp'suc:timeC";"1"] SUB_LESS_EQ)
          THEN IMP_RES_TAC LESS_EQ_TRANS
          THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LE_PRE_IMP_LT)
          THEN REWRITE_ASSUM_TAC
                 ("STABLE_FALSE_THEN_TRUE(Sack_Sig_Is_TRUE s' e')
                                         (ti',u)",
                  [STABLE_FALSE_THEN_TRUE])
```

```
              THEN POP_ASSUM_LIST (MAP_EVERY(\thm. STRIP_ASSUME_TAC thm))
              THEN IMP_RES_TAC NEW_P_RQT_FALSE_FROM_T'SACK_TO_TP'SUC
              THEN IMP_RES_TAC IBUS_ALE_IMP_NEW_P_RQT
              THEN RES_TAC
            ]
          ]
        ;
          % Subgoal 3.1.2.2.2: [ "~(ti' + 1) <= u" ] -contradiction- %
          IMP_RES_TAC NOT_LESS_EQ_LESS
          THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LT_SUC_IMP_LE)
          THEN IMP_RES_TAC LESS_EQUAL_ANTISYM
          THEN IMP_RES_TAC IBUS_ALE_IMP_PA
          THEN REWRITE_ASSUM_TAC
                 ("STABLE_FALSE_THEN_TRUE(Sack_Sig_Is_TRUE s' e')
                                         (ti',u)",
                  [STABLE_FALSE_THEN_TRUE])
          THEN POP_ASSUM_LIST (MAP_EVERY(\thm. STRIP_ASSUME_TAC thm))
          THEN SUBGOAL_THEN
                 "Sack_Sig_Is_TRUE s' e' u ==> New_State_Is_PD s' e' u"
                 IMP_RES_TAC
          THENL [
            % Subgoal 3.1.2.2.2.1:
              "Sack_Sig_Is_TRUE s' e' u ==> New_State_Is_PD s' e' u" %
              REWRITE_TAC [Sack_Sig_Is_TRUE]
              THEN BETA_TAC
              THEN REPEAT STRIP_TAC
              THEN ASM_REWRITE_TAC[]
          ;
            % Subgoal 3.1.2.2.2.2: (Continue) %
            REWRITE_ASSUM_TAC
              ("New_State_Is_PD s' e' u",[New_State_Is_PD])
            THEN UNDISCH_TAC "New_State_Is_PA s' e' ti'"
            THEN FILTER_ASM_REWRITE_TAC
                   (\tm. not (tm = "(u:timeC) = ti'"))
                   [New_State_Is_PA;
                    SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
          ]
        ]
      ]
    ]
  ;
    % Subgoal 3.2:  [ "~(?t'sack. Sack_Sig_Is_TRUE s' e' t'sack /\
                                  ti' <= t'sack /\
                                  t'sack <= tp'suc-1)" ] %
    UNDISCH_TAC "STABLE_FALSE_THEN_TRUE(ale_sig_pb e')(ti' + 1,tp'suc)"
    THEN REWRITE_TAC [STABLE_FALSE_THEN_TRUE]
    THEN BETA_TAC
    THEN REPEAT STRIP_TAC
    THEN SUBGOAL_THEN
       "STABLE_FALSE (Sack_Sig_Is_TRUE s' e') (ti',tp'suc-1)" ASSUME_TAC
    THENL [
       % Subgoal 3.2.1:
           "STABLE_FALSE(Sack_Sig_Is_TRUE s' e')(ti',tp'suc - 1)" %
       REWRITE_TAC [STABLE_FALSE]
       THEN BETA_TAC
       THEN IMP_RES_TAC LESS_LESS_EQ_TRANS
       THEN IMP_RES_TAC LT_IMP_LE
       THEN ASM_REWRITE_TAC[]
       THEN GEN_TAC
       THEN UNDISCH_TAC "~(?t'sack. Sack_Sig_Is_TRUE s' e' t'sack /\
                              ti' <= t'sack /\
                              t'sack <= (tp'suc-1))"
       THEN REWRITE_TAC
              [NOT_EXISTS_CONV "~(?t'sack. Sack_Sig_Is_TRUE s' e' t'sack /\
                                ti' <= t'sack /\ t'sack <= (tp'suc-1))";
                                DE_MORGAN_THM]
       THEN QUANT_OUT_IMP_TAC
       THEN EXISTS_TAC "t'':timeC"
       THEN REPEAT STRIP_TAC
       THEN RES_TAC
    ;
       % Subgoal 3.2.2:
```

```
                    [ "STABLE_FALSE(Sack_Sig_Is_TRUE s' e')(ti',tp'suc - 1)" ] %
               UNDISCH_TAC "ale_sig_ib p' t'"
               THEN UNDISCH_TAC "~(?t'sack. Sack_Sig_Is_TRUE s' e' t'sack /\
                                   ti' <= t'sack /\
                                   t'sack <= (tp'suc-1))"
               THEN REWRITE_TAC
                      [NOT_EXISTS_CONV "~(?t'sack. Sack_Sig_Is_TRUE s' e' t'sack /\
                                    ti' <= t'sack /\ t'sack <= (tp'suc-1))";
                                    DE_MORGAN_THM]
               THEN QUANT_OUT_IMP_TAC
               THEN EXISTS_TAC "t':timeC"
               THEN SUBGOAL_THEN
                      "~ti' <= t' \/ ~t' <= (tp'suc-1) =
                       ~(ti' <= t' /\ t' <= (tp'suc-1))"
                      (\thm. REWRITE_TAC [thm])
               THENL [
                  % Subgoal 3.2.2.1: -New subgoal-
                    "~ti' <= t'sack \/ ~t'sack <= (tp'suc-1) =
                     ~(ti' <= t'sack /\ t'sack <= (tp'suc-1))" %
                  BOOL_CASES_TAC "ti' <= t'"
                  THEN BOOL_CASES_TAC "t' <= (tp'suc-1)"
                  THEN ASM_REWRITE_TAC[]
               ;
                  % Subgoal 3.2.2.2 %
                  SUBGOAL_THEN
                    "!a b. a \/ b = b \/ a" (\thm. PURE_ONCE_REWRITE_TAC[thm])
                  THENL [
                    % Subgoal 3.2.2.2.1: "!a b. a \/ b = b \/ a" %
                    REPEAT GEN_TAC
                    THEN BOOL_CASES_TAC "a:bool"
                    THEN BOOL_CASES_TAC "b:bool"
                    THEN ASM_REWRITE_TAC[]
                  ;
                    % Subgoal 3.2.2.2.2 %
                    REWRITE_TAC [SYM_RULE IMP_DISJ_THM]
                    THEN REPEAT STRIP_TAC
                    THEN SUBGOAL_THEN
                           "!t. NTH_TIME_TRUE t(ale_sig_ib p')0 ti' ==>
                                  ale_sig_ib p' ti'" IMP_RES_TAC
                    THENL [
                      % Subgoal 3.2.2.2.2.1: (New subgoal) %
                      INDUCT_TAC
                      THEN REWRITE_TAC [NTH_TIME_TRUE;STABLE_FALSE_THEN_TRUE;
                                        ZERO_LESS_EQ]
                      THEN REPEAT STRIP_TAC
                      THEN ASM_REWRITE_TAC[]
                    ;
                      % Subgoal 3.2.2.2.2.2 (Continue) %
                        IMP_RES_TAC (REWRITE_RULE [ADD1] LT_IMP_SUC_LE)
                        THEN ASSUME_TAC (SPECL ["tp'suc:timeC";"1"] SUB_LESS_EQ)
                        THEN IMP_RES_TAC LESS_EQ_TRANS
                        THEN IMP_RES_TAC NEW_STATE_PD_FROM_TI'_TO_T'SACK_1
                        THEN UNDISCH_TAC "New_State_Is_PD s' e' t'"
                        THEN IMP_RES_TAC
                               (REWRITE_RULE [New_State_Is_PA] IBUS_ALE_IMP_PA)
                        THEN ASM_REWRITE_TAC
                               [New_State_Is_PD;
                                prove_constructors_distinct pfsm_ty_Axiom]
                    ]
                  ]
               ]
            ]
         ]
      ]
     );;

let NTH_TIME_TRUE_X_IMP_X = TAC_PROOF
    (([],
     "! (n :num) (t :time) (x :time->bool) .
      NTH_TIME_TRUE n x 0 t ==> x t"),
     INDUCT_TAC
     THEN REWRITE_TAC [NTH_TIME_TRUE;STABLE_FALSE_THEN_TRUE]
    THEN REPEAT STRIP_TAC
    THEN RES_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let NTH_TIME_FALSE_X_IMP_NOT_X = TAC_PROOF
   (([],
    "! (n :num) (t :time) (x :time->bool) .
     NTH_TIME_FALSE n x t0 t ==> ~x t"),
    INDUCT_TAC
    THEN REWRITE_TAC [NTH_TIME_FALSE;STABLE_TRUE_THEN_FALSE]
    THEN REPEAT STRIP_TAC
    THEN RES_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let TRANS_ONE_TO_ONE = TAC_PROOF
   (([],
    "! (ti' tp'suc :timeC) (t :timeT)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out) .
     PCSet_Correct s' e' p' ==>
      (tp' > 0) ==>
        PT_Exec pti s e p t ==>
         PT_PreC pti s e p t ==>
          PStateAbs pti s e p t s' e' p' tp' ==>
           PB_Slave pti e p t e' p' tp' ==>
            IBA_PMaster pti e p t e' p' ==>
             Rst_Slave pti e t e' ==>
              NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
               STABLE_FALSE_THEN_TRUE (ale_sig_ib p') (tp',ti') ==>
                TRUE_THEN_STABLE_FALSE (ale_sig_pb e') (tp',ti')"),
    REWRITE_TAC [STABLE_FALSE_THEN_TRUE;TRUE_THEN_STABLE_FALSE]
    THEN BETA_TAC
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC NEW_STATE_PD_FALSE_FROM_TP'_TO_TI'
    THEN IMP_RES_TAC NTH_TIME_TRUE_X_IMP_X
    THEN ASM_REWRITE_TAC[]
    THEN NRULE_ASSUM_TAC
           ("!ti'. STABLE_FALSE(ale_sig_ib p')(tp',ti' - 1) ==>
                   (!t'. tp' <= t' ==> t' <= ti' ==> ~New_State_Is_PD s' e' t')",
            ((REWRITE_RULE [STABLE_FALSE]) o (SPEC "ti':timeC")))
    THEN IMP_RES_TAC LESS_LESS_EQ_TRANS
    THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
    THEN IMP_RES_TAC LT_IMP_LE
    THEN IMP_RES_TAC LESS_EQ_TRANS
    THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LT_IMP_LE_PRE)
    THEN RES_TAC
    THEN SUBGOAL_THEN
           "(!t. tp' <= t /\ t <= (ti' - 1) ==> ~ale_sig_ib p' t)" ASSUME_TAC
    THENL [
       % Subgoal 1: "!t. tp' <= t /\ t <= (ti' - 1) ==> ~ale_sig_ib p' t" %
       GEN_TAC
       THEN SPEC_ASSUM_TAC
              ("!t. tp' <= t /\ t < ti' ==> ~ale_sig_ib p' t","t'':timeC")
       THEN ASM_CASES_TAC "tp' <= t''"
       THEN ASM_CASES_TAC "t'' <= (ti'-1)"
       THEN ASM_REWRITE_TAC[]
       THEN IMP_RES_TAC
              (REWRITE_RULE
                [PRE_SUB1] (SPECL ["t'':timeC";"ti':timeC"] LE_PRE_IMP_LT))
       THEN RES_TAC
    ;
       % Subgoal 2: (Continue) %
       ASM_REWRITE_ASSUM_TAC
         ("(!t. tp' <= t /\ t <= (ti' - 1) ==> ~ale_sig_ib p' t) ==>
           (!t'. tp' <= t' ==> t' <= ti' ==> ~New_State_Is_PD s' e' t')",[])
       THEN SUBGOAL_THEN
              "LESS_THAN_N_TIMES_FALSE 0 (bsig L_ready_O p') tp' ti'" ASSUME_TAC
       THENL [
         % Subgoal 2.1: (New subgoal) %
         REWRITE_TAC [LESS_THAN_N_TIMES_FALSE;STABLE_TRUE;bsig;BSel]
```

```
          THEN BETA_TAC
          THEN ASM_REWRITE_TAC[]
          THEN GEN_TAC
          THEN SPEC_ASSUM_TAC
                 ("!t'. tp' <= t' ==> t' <= ti' ==> ~New_State_Is_PD s' e' t'",
                  "t'':timeC")
          THEN ASM_CASES_TAC "tp' <= t'' /\ t'' <= ti'"
          THEN ASM_REWRITE_TAC[]
          THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
          THEN RES_TAC
          THEN IMP_RES_TAC L_ready_ISO
          THEN REWRITE_ASSUM_TAC ("~New_State_Is_PD s' e' t''",[New_State_Is_PD])
          THEN ASM_REWRITE_TAC[]
       ;
          % Subgoal 2.2: (Continue) %
          SUBGOAL_THEN
            "!n. LESS_THAN_N_TIMES_FALSE 0(bsig L_ready_O p')tp' ti' ==>
                 LESS_THAN_N_TIMES_FALSE n(bsig L_ready_O p')tp' ti'"
            IMP_RES_TAC
          THENL [
            % Subgoal 2.2.1: (New subgoal) %
            INDUCT_TAC
            THEN DISCH_TAC
            THEN RES_TAC
            THEN ASM_REWRITE_TAC[LESS_THAN_N_TIMES_FALSE]
          ;
            % Subgoal 2.2.2: (Continue) %
            IMP_RES_TAC PB_REQUEST_ASSUMPS
            THEN SPEC_ASSUM_TAC
                   ("!n. LESS_THAN_N_TIMES_FALSE n(bsig L_ready_O p')tp' ti'",
                    "VAL 1(SUBARRAY(BSel(L_ad_inE(e' (tp':timeC))))(1,0))")
            THEN RES_TAC
            THEN REWRITE_ASSUM_TAC
                   ("STABLE_FALSE(ale_sig_pb e')(tp' + 1,ti'+1)",[STABLE_FALSE])
            THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
            THEN SPEC_ASSUM_TAC
                   ("!t. (tp' + 1) <= t /\ t <= (ti' + 1) ==> ~ale_sig_pb e' t",
                    "t':timeC")
            THEN ASSUME_TAC (SPECL ["ti':timeC";"1"] LESS_EQ_ADD)
            THEN IMP_RES_TAC LESS_EQ_TRANS
            THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LT_IMP_SUC_LE)
            THEN RES_TAC
          ]
       ]
    ]
   );;

let PRIOR_EVENTS_EXIST = TAC_PROOF
   (([],
    "! (n m :num) (x :time->bool) (t :time) .
     NTH_TIME_TRUE n x 0 t ==>
       (m < n) ==>
         (?t'. t' < t /\ NTH_TIME_TRUE m x 0 t')"),
    INDUCT_TAC
    THENL [
       % Subgoal 1: (Base Case) %
       REPEAT STRIP_TAC
       THEN ASSUME_TAC (SPEC "m:num" ZERO_LESS_EQ)
       THEN IMP_RES_TAC LESS_EQ_ANTISYM
    ;
       % Subgoal 2: (Induction Step) %
       REWRITE_TAC [NTH_TIME_TRUE]
       THEN REPEAT STRIP_TAC
       THEN RES_TAC
       THEN REWRITE_ASSUM_TAC
              ("STABLE_FALSE_THEN_TRUE x(t' + 1,t)",[STABLE_FALSE_THEN_TRUE])
       THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
       THEN IMP_RES_TAC (REWRITE_RULE [ADD1] SUC_LE_IMP_LT)
       THEN ASM_CASES_TAC "(m:num) = n"
       THENL [
          % Subgoal 2.1: [ "m = n" ] %
          EXISTS_TAC "t':time"
```

```
            THEN ASM_REWRITE_TAC[]
         ;
            % Subgoal 2.2: [ "~(m = n)" ] %
            IMP_RES_TAC NOT_EQ
            THENL [
               % Subgoal 2.2.1: [ "m < n" ] %
               IMP_RES_TAC LESS_EQ_ANTISYM
               THEN RES_TAC
               THEN EXISTS_TAC "t'':time"
               THEN IMP_RES_TAC LESS_TRANS
               THEN ASM_REWRITE_TAC[]
            ;
               % Subgoal 2.2.2: [ "n < m" ] %
               IMP_RES_TAC LT_SUC_IMP_LE
               THEN IMP_RES_TAC LESS_EQ_ANTISYM
            ]
         ]
      ]
   );;

let NTH_TRANS_CAUSAL = TAC_PROOF
   (([],
   "! (t :timeT) (tp' ti' :timeC) (e' :timeC->pc_env) (p' :timeC->pc_out)
      (pti :PTI) .
    PCSet_Correct s' e' p' ==>
     PTAbsSet s e p s' e' p' ==>
      (tp' > 0) ==>
        PT_Exec pti s e p t ==>
         PT_PreC pti s e p t ==>
          PStateAbs pti s e p t s' e' p' tp' ==>
           IBA_PMaster pti e p t e' p' ==>
            Rst_Slave pti e t e' ==>
             NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
              NTH_TIME_TRUE t (ale_sig_ib p') 0 ti' ==>
               (tp' <= ti')"),
    INDUCT_TAC
    THENL [
       % Subgoal 1: (Base Case) %
       REPEAT STRIP_TAC
       THEN ASM_CASES_TAC "tp' <= ti'"
       THEN ASM_REWRITE_TAC[]
       THEN IMP_RES_TAC NOT_LESS_EQ_LESS
       THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
       THEN IMP_RES_TAC ALE_SIG_IB_FALSE_UPTO_FIRST
       THEN UNDISCH_TAC "NTH_TIME_TRUE 0(ale_sig_ib p')0 ti'"
       THEN REWRITE_TAC [NTH_TIME_TRUE;STABLE_FALSE_THEN_TRUE;DE_MORGAN_THM]
       THEN REWRITE_ASSUM_TAC
              ("STABLE_FALSE(ale_sig_ib p')(0,tp' - 1)",[STABLE_FALSE])
       THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
       THEN SPEC_ASSUM_TAC
              ("!t. 0 <= t /\ t <= (tp' - 1) ==> ~ale_sig_ib p' t","ti':timeC")
       THEN ASSUME_TAC (SPEC "ti':timeC" ZERO_LESS_EQ)
       THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LT_IMP_LE_PRE)
       THEN RES_TAC
       THEN ASM_REWRITE_TAC[]
    ;
       % Subgoal 2: (Induction Step) %
       REPEAT STRIP_TAC
       THEN IMP_RES_TAC ABS_SET_IMP_ABS
       THEN IMP_RES_TAC PRE_EXEC_PREC
       THEN NRULE_ASSUM_TAC
              ("!pti t. PTAbs pti s e p t s' e' p'",
               ((REWRITE_RULE [PTAbs]) o (SPECL ["pti0:PTI";"t:timeT"])))
       THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
       THEN RES_TAC
       THEN RES_TAC
       THEN ASSUME_TAC (SPEC "t:timeT" LESS_SUC_REFL)
       THEN IMP_RES_TAC PRIOR_EVENTS_EXIST
       THEN RES_TAC
       THEN IMP_RES_TAC TRANS_ONTO
       THEN SPEC_ASSUM_TAC
             ("!tp'suc. STABLE_FALSE_THEN_TRUE(ale_sig_pb e')(t'' + 1,tp'suc) ==>
              TRUE_THEN_STABLE_FALSE(ale_sig_ib p')(t'',tp'suc - 1)","tp':timeC")
       THEN SUBGOAL_THEN
              "STABLE_FALSE_THEN_TRUE(ale_sig_pb e')(tp'' + 1,tp')"
              ASSUME_TAC
       THENL [
         % Subgoal 2.1: (New Subgoal) %
         REWRITE_ASSUM_TAC
                ("NTH_TIME_TRUE(SUC t)(ale_sig_pb e')0 tp'",[NTH_TIME_TRUE])
         THEN CHOOSE_ASSUM_TAC
                "?t'. NTH_TIME_TRUE t(ale_sig_pb e')0 t' /\
                      STABLE_FALSE_THEN_TRUE(ale_sig_pb e')(t' + 1,tp')"
         THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
         THEN SUBGOAL_THEN "(tp'':timeC) = t'''" (\thm. ASM_REWRITE_TAC [thm])
         THEN IMP_RES_TAC TRUE_EVENT_TIMES_EQUAL
       ;
         % Subgoal 2.2: (Continue) %
         ASM_CASES_TAC "(t''+1) <= tp'"
         THENL [
           % Subgoal 2.2.1: [ "(t'' + 1) <= tp'" ] %
           IMP_RES_TAC
             (RIMP (SPECL ["tp'':timeC";"t'':timeC";"1"] LESS_EQ_MONO_ADD_EQ))
           THEN IMP_RES_TAC SUB_STABLE_FALSE_THEN_TRUE
           THEN RES_TAC
           THEN ASM_CASES_TAC "tp' <= ti'"
           THEN ASM_REWRITE_TAC[]
           THEN IMP_RES_TAC NOT_LESS_EQ_LESS
           THEN IMP_RES_TAC NTH_TIME_TRUE_X_IMP_X
           THEN NRULE_ASSUM_TAC
                  ("TRUE_THEN_STABLE_FALSE(ale_sig_ib p')(t'',tp' - 1)",
                   (BETA_RULE o (REWRITE_RULE [TRUE_THEN_STABLE_FALSE])))
           THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
           THEN SPEC_ASSUM_TAC
                  ("!t. t'' < t /\ t <= (tp'-1) ==> ~ale_sig_ib p' t","ti':timeC")
           THEN IMP_RES_TAC TRUE_EVENT_TIMES_MONO
           THEN IMP_RES_TAC LT_IMP_LE
           THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LT_IMP_LE_PRE)
           THEN RES_TAC
         ;
           % Subgoal 2.2.2: [ "~(t'' + 1) <= tp'" ] %
           IMP_RES_TAC NOT_LESS_EQ_LESS
           THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LT_SUC_IMP_LE)
           THEN IMP_RES_TAC LESS_EQ_LESS_TRANS
           THEN IMP_RES_TAC LT_IMP_LE
           THEN ASM_REWRITE_TAC[]
         ]
       ]
    ]
   );;

let NTH_TRANS_ONE_TO_ONE = TAC_PROOF
   (([],
   "! (t :timeT) (tp' ti' :timeC)
      (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
      (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
      (pti :PTI) .
    PCSet_Correct s' e' p' ==>
     PTAbsSet s e p s' e' p' ==>
      (tp' > 0) ==>
        PT_Exec pti s e p t ==>
         PT_PreC pti s e p t ==>
          PStateAbs pti s e p t s' e' p' tp' ==>
           PB_Slave pti e p t e' p' tp' ==>
            IBA_PMaster pti e p t e' p' ==>
             Rst_Slave pti e t e' ==>
              NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
               NTH_TIME_TRUE t (ale_sig_ib p') 0 ti' ==>
                TRUE_THEN_STABLE_FALSE (ale_sig_pb e') (tp',ti')"),
    INDUCT_TAC
    THENL [
       % Subgoal 1: (Base Case) %
       REPEAT STRIP_TAC
       THEN IMP_RES_TAC NTH_TRANS_CAUSAL
```

```
        THEN IMP_RES_TAC TRANS_ONE_TO_ONE
        THEN SPEC_ASSUM_TAC
               ("!ti'. STABLE_FALSE_THEN_TRUE(ale_sig_ib p')(tp',ti') ==>
                 TRUE_THEN_STABLE_FALSE(ale_sig_pb e')(tp',ti')","ti':timeC")
        THEN REWRITE_ASSUM_TAC
               ("NTH_TIME_TRUE 0(ale_sig_ib p')0 ti'",[NTH_TIME_TRUE])
        THEN IMP_RES_TAC GREATER
        THEN IMP_RES_TAC LT_IMP_LE
        THEN IMP_RES_TAC SUB_STABLE_FALSE_THEN_TRUE
        THEN RES_TAC
     ;
        % Subgoal 2: (Induction Step) %
        REPEAT STRIP_TAC
        THEN IMP_RES_TAC ABS_SET_IMP_ABS
        THEN IMP_RES_TAC PRE_EXEC_PREC
        THEN NRULE_ASSUM_TAC
               ("!pti t. PTAbs pti s e p t s' e' p'",
                ((REWRITE_RULE [PTAbs]) o (SPECL ["pti0:PTI";"t:timeT"])))
        THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
        THEN RES_TAC
        THEN RES_TAC
        THEN ASSUME_TAC (SPEC "t:timeT" LESS_SUC_REFL)
        THEN IMP_RES_TAC PRIOR_EVENTS_EXIST
        THEN RES_TAC
        THEN IMP_RES_TAC TRANS_ONE_TO_ONE
        THEN SPEC_ASSUM_TAC
               ("!ti'. STABLE_FALSE_THEN_TRUE(ale_sig_ib p')(tp',ti') ==>
                 TRUE_THEN_STABLE_FALSE(ale_sig_pb e')(tp',ti')","ti':timeC")
        THEN ASM_CASES_TAC "t'' < tp'"
        THENL [
          % Subgoal 2.1: [ "t'' < tp'" ] %
          IMP_RES_TAC LT_IMP_LE
          THEN IMP_RES_TAC NTH_TRANS_CAUSAL
          THEN REWRITE_ASSUM_TAC
                 ("NTH_TIME_TRUE(SUC t)(ale_sig_ib p')0 ti'",[NTH_TIME_TRUE])
          THEN CHOOSE_ASSUM_TAC
                 "?t'. NTH_TIME_TRUE t(ale_sig_ib p')0 t' /\
                       STABLE_FALSE_THEN_TRUE(ale_sig_ib p')(t' + 1,ti')"
          THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
          THEN SUBGOAL_THEN "(t''':timeC) = t''" ASSUME_TAC
          THENL [
            % Subgoal 2.1.1: (New subgoal) %
            IMP_RES_TAC TRUE_EVENT_TIMES_EQUAL
          ;
            % Subgoal 2.1.2: (Continue) %
            ASM_REWRITE_ASSUM_TAC
              ("STABLE_FALSE_THEN_TRUE(ale_sig_ib p')(t''' + 1,ti')",[])
            THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LT_IMP_SUC_LE)
            THEN IMP_RES_TAC SUB_STABLE_FALSE_THEN_TRUE
            THEN RES_TAC
          ]
        ;
          % Subgoal 2.2: [ "~t'' < tp'" ] -contradiction- %
          IMP_RES_TAC NOT_LESS
          THEN IMP_RES_TAC TRUE_EVENT_TIMES_MONO
          THEN IMP_RES_TAC NTH_TIME_TRUE_X_IMP_X
          THEN REWRITE_ASSUM_TAC
                 ("TRUE_THEN_STABLE_FALSE(ale_sig_pb e')(tp'',t'')",
                  [TRUE_THEN_STABLE_FALSE])
          THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
          THEN SPEC_ASSUM_TAC
                 ("!t. tp'' < t /\ t <= t'' ==> ~ale_sig_pb e' t","tp':timeC")
          THEN RES_TAC

        ]
     ]
    );;

let NTH_TRANS_ONTO = TAC_PROOF
   (([],
    "! (t :timeT) (tp' tp'' ti'' :timeC)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
```

```
         (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
         (pti :PTI) .
      PCSet_Correct s' e' p' ==>
       PTAbsSet s e p s' e' p' ==>
         (tp'' > 0) ==>
           PT_Exec pti s e p t ==>
            PT_PreC pti s e p t ==>
             PStateAbs pti s e p t s' e' p' tp'' ==>
              PB_Slave pti e p t e' p' tp'' ==>
               IBA_PMaster pti e p t e' p' ==>
                Rst_Slave pti e t e' ==>
                 NTH_TIME_TRUE t (ale_sig_ib p') 0 ti'' ==>
                  NTH_TIME_TRUE t (ale_sig_pb e') 0 tp'' ==>
                   NTH_TIME_TRUE (SUC t) (ale_sig_pb e') 0 tp' ==>
                    TRUE_THEN_STABLE_FALSE (ale_sig_ib p') (ti'',tp'-1)"),
      REPEAT STRIP_TAC
      THEN IMP_RES_TAC TRANS_ONTO
      THEN SPEC_ASSUM_TAC
             ("!tp'suc. STABLE_FALSE_THEN_TRUE(ale_sig_pb e')(ti''+1,tp'suc) ==>
                TRUE_THEN_STABLE_FALSE(ale_sig_ib p')(ti'',tp'suc-1)","tp':timeC")
      THEN IMP_RES_TAC NTH_TRANS_CAUSAL
      THEN ASSUME_TAC (SPECL ["ti'':timeC";"1"] LESS_EQ_ADD)
      THEN IMP_RES_TAC LESS_EQ_TRANS
      THEN IMP_RES_TAC TRUE_EVENT_TIMES_MONO
      THEN IMP_RES_TAC NTH_TIME_TRUE_X_IMP_X
      THEN REWRITE_ASSUM_TAC
              ("NTH_TIME_TRUE(SUC t)(ale_sig_pb e')0 tp'",[NTH_TIME_TRUE])
      THEN CHOOSE_ASSUM_TAC
              "?t'. NTH_TIME_TRUE t(ale_sig_pb e')0 t' /\
                    STABLE_FALSE_THEN_TRUE(ale_sig_pb e')(t' + 1,tp')"
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN SUBGOAL_THEN "(t':timeC) = tp''" ASSUME_TAC
      THENL [
         % Subgoal 1: (New subgoal) %
         IMP_RES_TAC TRUE_EVENT_TIMES_EQUAL
      ;
         % Subgoal 2: (Continue) %
         ASM_REWRITE_ASSUM_TAC
           ("STABLE_FALSE_THEN_TRUE(ale_sig_pb e')(t' + 1,tp')",[])
         THEN ASM_CASES_TAC "(ti'' + 1) <= tp'"
         THENL [
           % Subgoal 2.1: [ "(ti'' + 1) <= tp'" ] %
           IMP_RES_TAC
             (RIMP (SPECL ["tp'':timeC";"ti'':timeC";"1"] LESS_EQ_MONO_ADD_EQ))
           THEN IMP_RES_TAC SUB_STABLE_FALSE_THEN_TRUE
           THEN RES_TAC
         ;
           % Subgoal 2.2: [ "~(ti'' + 1) <= tp'" ] %
           IMP_RES_TAC NOT_LESS_EQ_LESS
           THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LT_SUC_IMP_LE)
           THEN IMP_RES_TAC NTH_TRANS_ONE_TO_ONE
           THEN REWRITE_ASSUM_TAC
                  ("TRUE_THEN_STABLE_FALSE(ale_sig_pb e')(tp'',ti'')",
                   [TRUE_THEN_STABLE_FALSE])
           THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
           THEN SPEC_ASSUM_TAC
                  ("!t. tp'' < t /\ t <= ti'' ==> ~ale_sig_pb e' t","tp':timeC")
           THEN IMP_RES_TAC TRUE_EVENT_TIMES_MONO
           THEN RES_TAC
         ]
      ]
     );;

let NTH_IBUS_TRANS_EXISTS = TAC_PROOF
   (([],
    "! (t :timeT) (pti :PTI)
       (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' :timeC) .
   PCSet_Correct s' e' p' ==>
     PTAbsSet s e p s' e' p' ==>
       PT_Exec pti s e p t ==>
     PT_PreC pti s e p t ==>
       (?ti':timeC. NTH_TIME_TRUE t (ale_sig_ib p') 0 ti' /\ ti' > 0)"),
INDUCT_TAC
THEN REPEAT STRIP_TAC
THENL [
   % Subgoal 1: (Base Case) %
   IMP_RES_TAC ABS_SET_IMP_ABS
   THEN NRULE_ASSUM_TAC
          ("!pti t. PTAbs pti s e p t s' e' p'",
           ((REWRITE_RULE [PTAbs]) o (SPECL ["pti:PTI";"0"])))
   THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
   THEN RES_TAC
   THEN RES_TAC
   THEN ASM_CASES_TAC
          "~ELEMENT (FST(L_ad_inE (e' tp'))) 31 /\ New_State_Is_PA s' e' tp'"
   THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
   THEN PURE_ONCE_REWRITE_ASSUM_TAC
          ("1 <= tp'",[SYM_RULE (REDUCE_CONV "0+1")])
   THEN IMP_RES_TAC SUC_LE_IMP_LE
   THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
   THENL [
      % Subgoal 1.1: "?ti'. NTH_TIME_TRUE 0(ale_sig_ib p')0 ti' /\ ti' > 0"
                     [ "~ELEMENT(FST(L_ad_inE(e' tp')))31 /\
                        New_State_Is_PA s' e' tp'" ] %
      POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN IMP_RES_TAC ALE_SIG_IB_TRUE_ON_TP'
      THEN EXISTS_TAC "tp':timeC"
      THEN ASM_REWRITE_TAC [NTH_TIME_TRUE;STABLE_FALSE_THEN_TRUE]
      THEN IMP_RES_TAC ALE_SIG_IB_FALSE_UPTO_FIRST
      THEN NRULE_ASSUM_TAC
             ("STABLE_FALSE(ale_sig_ib p')(0,tp' - 1)",
              (BETA_RULE o (REWRITE_RULE [STABLE_FALSE])))
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN GEN_TAC
      THEN ASSUME_TAC
             (SPECL ["t:timeC";"tp':timeC"]
                    (REWRITE_RULE [PRE_SUB1] LT_EQ_LE_PRE))
      THEN RES_TAC
      THEN ASM_REWRITE_TAC[]
   ;
      % Subgoal 1.2: "?ti'. NTH_TIME_TRUE 0(ale_sig_ib p')0 ti' /\ ti' > 0"
                     [ "~(~ELEMENT(FST(L_ad_inE(e' tp')))31 /\
                          New_State_Is_PA s' e' tp')" ] %

      IMP_RES_TAC ALE_SIG_IB_TRUE_AFTER_TP'
      THEN IMP_RES_TAC ALE_SIG_IB_FALSE_UPTO_FIRST
      THEN IMP_RES_TAC SUP_INTERVAL_STABLE_FALSE_THEN_TRUE
      THEN IMP_RES_TAC (SPECL ["tp':timeC";"1"] SUB_ADD)
      THEN ASSUME_TAC (SPEC "tp':timeC" LESS_EQ_REFL)
      THEN REWRITE_ASSUM_TAC
             ("STABLE_FALSE_THEN_TRUE(ale_sig_ib p')(tp',ti')",
              [STABLE_FALSE_THEN_TRUE])
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN IMP_RES_TAC LESS_EQ_TRANS
      THEN ASM_REWRITE_ASSUM_TAC
             ("tp' <= ((tp' - 1) + 1) ==>
                  0 <= ti' ==>
                    STABLE_FALSE_THEN_TRUE(ale_sig_ib p')(0,ti')",[])
      THEN IMP_RES_TAC (LIMP ONE_LESS_EQ)
      THEN EXISTS_TAC "ti':timeC"
      THEN ASM_REWRITE_TAC [NTH_TIME_TRUE]
   ]
;
   % Subgoal 2: (Induction Step) %
   IMP_RES_TAC PRE_EXEC_PREC
   THEN RES_TAC
   THEN IMP_RES_TAC ABS_SET_IMP_ABS
   THEN NRULE_ASSUM_TAC
          ("!pti t. PTAbs pti s e p t s' e' p'",
           ((REWRITE_RULE [PTAbs]) o (SPECL ["pti:PTI";"SUC t"])))
   THEN IMP_RES_TAC ABS_SET_IMP_ABS
   THEN NRULE_ASSUM_TAC
```

```
        ("!pti t. PTAbs pti s e p t s' e' p'",
         ((REWRITE_RULE [PTAbs]) o (SPECL ["pti0:PTI";"t:timeT"])))
THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
THEN RES_TAC
THEN RES_TAC
THEN ASM_CASES_TAC
        "~ELEMENT (FST(L_ad_inE (e' tp'))) 31 /\ New_State_Is_PA s' e' tp'"
THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
THEN PURE_ONCE_REWRITE_ASSUM_TAC
        ("1 <= tp'",[SYM_RULE (REDUCE_CONV "0+1")])
THEN IMP_RES_TAC SUC_LE_IMP_LE
THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
THENL [
   % Subgoal 2.1: "?ti'.NTH_TIME_TRUE(SUC t)(ale_sig_ib p')0 ti' /\ ti'>0"
                  [ "~ELEMENT(FST(L_ad_inE(e' tp')))31 /\
                     New_State_Is_PA s' e' tp'" ] %
   EXISTS_TAC "tp':timeC"
   THEN ASM_REWRITE_TAC [NTH_TIME_TRUE]
   THEN EXISTS_TAC "ti'':timeC"
   THEN ASM_REWRITE_TAC [STABLE_FALSE_THEN_TRUE]
   THEN IMP_RES_TAC NTH_TRANS_ONTO
   THEN NRULE_ASSUM_TAC
         ("TRUE_THEN_STABLE_FALSE(ale_sig_ib p')(ti'',tp' - 1)",
          (BETA_RULE o (REWRITE_RULE [TRUE_THEN_STABLE_FALSE])))
   THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
   THEN ASSUME_TAC
        (SPECL ["ti'':timeC";"tp'-1";"1"] (SYM_RULE LESS_EQ_MONO_ADD_EQ))
   THEN IMP_RES_TAC (SPECL ["tp':timeC";"1"] SUB_ADD)
   THEN ASM_REWRITE_ASSUM_TAC ("ti'' <= (tp' - 1)",[])
   THEN IMP_RES_TAC ALE_SIG_IB_TRUE_ON_TP'
   THEN ASM_REWRITE_TAC[]
   THEN GEN_TAC
   THEN SPEC_ASSUM_TAC
         ("!t. ti'' < t' /\ t' <= (tp' - 1) ==> ~ale_sig_ib p' t",
          "t':timeC")
   THEN IMP_RES_TAC
      (SPECL ["t':timeC";"tp':timeC"]
            (REWRITE_RULE [PRE_SUB1] LT_EQ_LE_PRE))
   THEN ASM_REWRITE_TAC
         [REWRITE_RULE [ADD1]
           (SPECL ["ti'':timeC";"t':timeC"] (SYM_RULE LT_EQ_SUC_LE))]
   THEN ASM_CASES_TAC "ti'' < t' /\ t' <= (tp' - 1)"
   THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
   THEN RES_TAC
   THEN ASM_REWRITE_TAC[]
;
% Subgoal 2.2: "?ti'. NTH_TIME_TRUE(SUC t)(ale_sig_ib p')0 ti' /\ ti' > 0"
               [ "~(~ELEMENT(FST(L_ad_inE(e' tp')))31 /\
                  New_State_Is_PA s' e' tp')" ] %
   REWRITE_TAC [NTH_TIME_TRUE;STABLE_FALSE_THEN_TRUE;SYM_RULE ONE_LESS_EQ]
   THEN IMP_RES_TAC ALE_SIG_IB_TRUE_AFTER_TP'
   THEN EXISTS_TAC "ti'''':timeC"
   THEN IMP_RES_TAC NTH_TRANS_ONTO
   THEN NRULE_ASSUM_TAC
         ("STABLE_FALSE_THEN_TRUE(ale_sig_ib p')(tp',ti'''')",
          (BETA_RULE o (REWRITE_RULE [STABLE_FALSE_THEN_TRUE])))
   THEN NRULE_ASSUM_TAC
         ("TRUE_THEN_STABLE_FALSE(ale_sig_ib p')(ti'',tp' - 1)",
          (BETA_RULE o (REWRITE_RULE [TRUE_THEN_STABLE_FALSE])))
   THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
   THEN IMP_RES_TAC (SPECL ["tp':timeC";"1"] SUB_ADD)
   THEN ASM_REWRITE_ASSUM_TAC
         ("ti'' <= (tp' - 1)",
          [SYM_RULE (SPECL ["ti'':timeC";"tp'-1";"1"]
                           LESS_EQ_MONO_ADD_EQ)])
   THEN IMP_RES_TAC LESS_EQ_TRANS
   THEN ASM_REWRITE_TAC []
   THEN EXISTS_TAC "ti'':timeC"
   THEN ASM_REWRITE_TAC []
   THEN GEN_TAC
   THEN ASM_CASES_TAC "(ti'' + 1) <= t' /\ t' < ti''''"
   THEN ASM_REWRITE_TAC[]
```

```
        THEN ASM_CASES_TAC "t' <= tp' - 1"
        THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
        THENL [
           % Subgoal 2.2.1: [ "t' <= (tp' - 1)" ] %
           SPEC_ASSUM_TAC
              ("!t. ti'' < t /\ t <= (tp'-1) ==> ~ale_sig_ib p' t","t':timeC")
           THEN ASSUME_TAC
                  (PURE_ONCE_REWRITE_RULE
                     [ADD_SYM] (SPECL ["1";"ti'':timeC"] LESS_EQ_ADD))
           THEN IMP_RES_TAC LESS_EQ_TRANS
           THEN IMP_RES_TAC
                  (SPECL ["ti'':timeC";"t':timeC"]
                         (REWRITE_RULE [ADD1] SUC_LE_IMP_LT))
           THEN RES_TAC
        ;
           % Subgoal 2.2.2: [ "~t' <= (tp' - 1)" ] %
           IMP_RES_TAC NOT_LESS_EQ_LESS
           THEN SPEC_ASSUM_TAC
                     ("!t. tp'<=t /\ t<ti'''' ==> ~ale_sig_ib p' t","t':timeC")
           THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] PRE_LT_IMP_LE)
           THEN RES_TAC
        ]
     ]
   ]
  );;

let TRANS_TIMES_EQUAL = TAC_PROOF
  (([],
   "! (t :timeT) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
      (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
      (tp' :timeC) (ti' :timeC) (pti :PTI) .
     PCSet_Correct s' e' p' ==>
      PTAbsSet s e p s' e' p' ==>
       NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
        tp' > 0 ==>
         NTH_TIME_TRUE t (ale_sig_ib p') 0 ti' ==>
           PT_Exec pti s e p t ==>
            PT_PreC pti s e p t ==>
             PStateAbs pti s e p t s' e' p' tp' ==>
              Rst_Slave pti e t e' ==>
               PB_Slave pti e p t e' p' tp' ==>
                IB_PMaster pti e p t e' p' ti' ==>
                 IBA_PMaster pti e p t e' p' ==>
                  (New_State_Is_PA s' e' tp' /\
                   ~ELEMENT (FST(L_ad_inE (e' tp'))) (31)) ==>
                     (ti' = tp')"),
   INDUCT_TAC
   THEN REWRITE_TAC [New_State_Is_PA]
   THEN REPEAT STRIP_TAC
   THEN ASSUME_TAC
          (SPECL ["t:timeT";"0";"ti':timeC";"tp':timeC";"ale_sig_ib p'"]
                 TRUE_EVENT_TIMES_EQUAL)
   THEN RES_TAC
   THEN IMP_RES_TAC NTH_TRANS_CAUSAL
   THEN IMP_RES_TAC LESS_OR_EQ
   THEN ASM_REWRITE_TAC[]
   THEN IMP_RES_TAC (REWRITE_RULE [New_State_Is_PA] ALE_SIG_IB_TRUE_ON_TP')
   THENL [
      % Subgoal 1: (Base Case) %
      REWRITE_ASSUM_TAC
        ("NTH_TIME_TRUE 0(ale_sig_ib p')0 ti'",
         [NTH_TIME_TRUE;STABLE_FALSE_THEN_TRUE])
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN SPEC_ASSUM_TAC
             ("!t. 0 <= t /\ t < ti' ==> ~ale_sig_ib p' t","tp':timeC")
      THEN IMP_RES_TAC GREATER
      THEN IMP_RES_TAC LT_IMP_LE
      THEN RES_TAC
   ;
      % Subgoal 2: (Induction Step) %
      REWRITE_ASSUM_TAC
             ("NTH_TIME_TRUE(SUC t)(ale_sig_ib p')0 ti'",
              [NTH_TIME_TRUE;STABLE_FALSE_THEN_TRUE])
        THEN CHOOSE_ASSUM_TAC
               "?t'. NTH_TIME_TRUE t(ale_sig_ib p')0 t' /\
                     (t' + 1) <= ti' /\
                     (!t. (t' + 1) <= t /\ t < ti' ==> ~ale_sig_ib p' t) /\
                     ale_sig_ib p' ti'"
        THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
        THEN SPEC_ASSUM_TAC
               ("!t''. (t' + 1) <= t'' /\ t'' < ti' ==> ~ale_sig_ib p' t''",
                "tp':timeC")
        THEN IMP_RES_TAC PRE_EXEC_PREC
        THEN IMP_RES_TAC ABS_SET_IMP_ABS
        THEN NRULE_ASSUM_TAC
               ("!pti t. PTAbs pti s e p t s' e' p'",
                ((REWRITE_RULE [PTAbs]) o (SPECL ["pti0:PTI";"t:timeT"])))
        THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
        THEN RES_TAC
        THEN RES_TAC
        THEN IMP_RES_TAC NTH_TRANS_ONTO
        THEN REWRITE_ASSUM_TAC
               ("TRUE_THEN_STABLE_FALSE(ale_sig_ib p')(t',tp' - 1)",
                [TRUE_THEN_STABLE_FALSE])
        THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
        THEN IMP_RES_TAC
               (SPECL ["t':timeC";"tp'-1";"1"] (RIMP LESS_EQ_MONO_ADD_EQ))
        THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
        THEN IMP_RES_TAC (SPECL ["tp':timeC";"1"] SUB_ADD)
        THEN ASM_REWRITE_ASSUM_TAC ("(t' + 1) <= ((tp' - 1) + 1)",[])
        THEN RES_TAC
     ]
    );;

let NEXT_IBUS_TRANS_IS_NTH = TAC_PROOF
   (([],
    "! (t :timeT) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' :timeC) (ti' :timeC) (pti :PTI) .
      PCSet_Correct s' e' p' ==>
       PTAbsSet s e p s' e' p' ==>
        NTH_TIME_TRUE t (ale_sig_pb e') 0 tp' ==>
         tp' > 0 ==>
           PT_Exec pti s e p t ==>
            PT_PreC pti s e p t ==>
             PStateAbs pti s e p t s' e' p' tp' ==>
              Rst_Slave pti e t e' ==>
               PB_Slave pti e p t e' p' tp' ==>
                IBA_PMaster pti e p t e' p' ==>
                 STABLE_FALSE_THEN_TRUE (ale_sig_ib p') (tp',ti') ==>
                  NTH_TIME_TRUE t (ale_sig_ib p') 0 ti'"),
    INDUCT_TAC
    THEN REPEAT STRIP_TAC
    THENL [
        % Subgoal 1: (Base Case) %
        REPEAT STRIP_TAC
        THEN IMP_RES_TAC ALE_SIG_IB_FALSE_UPTO_FIRST
        THEN IMP_RES_TAC SUP_INTERVAL_STABLE_FALSE_THEN_TRUE
        THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
        THEN IMP_RES_TAC (SPECL ["tp':timeC";"1"] SUB_ADD)
        THEN ASSUME_TAC (SPEC "ti':timeC" ZERO_LESS_EQ)
        THEN ASSUME_TAC (SPEC "tp':timeC" LESS_EQ_REFL)
        THEN ASM_REWRITE_ASSUM_TAC
               ("tp' <= ((tp' - 1) + 1) ==> 0 <= ti' ==>
                  STABLE_FALSE_THEN_TRUE(ale_sig_ib p')(0,ti')",[])
        THEN ASM_REWRITE_TAC [NTH_TIME_TRUE]
     ;
        % Subgoal 2: (Induction Step) %
        IMP_RES_TAC PRE_EXEC_PREC
        THEN IMP_RES_TAC ABS_SET_IMP_ABS
        THEN NRULE_ASSUM_TAC
               ("!pti t. PTAbs pti s e p t s' e' p'",
                ((REWRITE_RULE [PTAbs]) o (SPECL ["pti0:PTI";"t:timeT"])))
        THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN RES_TAC
      THEN RES_TAC
      THEN POP_ASSUM (\thm. ALL_TAC)
      THEN POP_ASSUM (\thm. ALL_TAC)
      THEN IMP_RES_TAC NTH_IBUS_TRANS_EXISTS
      THEN IMP_RES_TAC NTH_TRANS_ONTO
      THEN POP_ASSUM (\thm. ALL_TAC)
      THEN POP_ASSUM (\thm. ALL_TAC)
      THEN REWRITE_TAC [NTH_TIME_TRUE;STABLE_FALSE_THEN_TRUE]
      THEN EXISTS_TAC "ti''':timeC"
      THEN ASM_REWRITE_TAC[]
      THEN REWRITE_ASSUM_TAC
              ("TRUE_THEN_STABLE_FALSE(ale_sig_ib p')(ti''',tp' - 1)",
               [TRUE_THEN_STABLE_FALSE])
      THEN REWRITE_ASSUM_TAC
              ("STABLE_FALSE_THEN_TRUE(ale_sig_ib p')(tp',ti')",
               [STABLE_FALSE_THEN_TRUE])
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN IMP_RES_TAC
              (RIMP (SPECL ["ti''':timeC";"tp'-1";"1"] LESS_EQ_MONO_ADD_EQ))
      THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
      THEN IMP_RES_TAC (SPECL ["tp':timeC";"1"] SUB_ADD)
      THEN ASM_REWRITE_ASSUM_TAC ("(ti''' + 1) <= ((tp' - 1) + 1)",[])
      THEN IMP_RES_TAC LESS_EQ_TRANS
      THEN ASM_REWRITE_TAC[]
      THEN REPEAT STRIP_TAC
      THEN SPEC_ASSUM_TAC
              ("!t. tp' <= t /\ t < ti' ==> ~ale_sig_ib p' t","t':timeC")
      THEN SPEC_ASSUM_TAC
              ("!t. ti''' < t /\ t <= (tp' - 1) ==> ~ale_sig_ib p' t","t':timeC")
      THEN ASM_CASES_TAC "tp' <= t'"
      THENL [
         % Subgoal 2.1: [ "tp' <= t'" ] %
         RES_TAC
      ;
         % Subgoal 2.2: [ "~tp' <= t'" ] %
         IMP_RES_TAC NOT_LESS_EQ_LESS
         THEN IMP_RES_TAC (REWRITE_RULE [ADD1] SUC_LE_IMP_LT)
         THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LT_IMP_LE_PRE)
         THEN RES_TAC
      ]
   ]
  );;

let P_RQT_FALSE_ON_TI' = TAC_PROOF
  (([“PCSet_Correct s' e' p'";
     "PTAbsSet s e p s' e' p'";
     "NTH_TIME_TRUE t (ale_sig_pb e') 0 tp'";
     "tp' > 0";
     "NTH_TIME_TRUE t (ale_sig_ib p') 0 ti'";
     "PT_Exec pti s e p t";
     "PT_PreC pti s e p t";
     "PStateAbs pti s e p t s' e' p' tp'";
     "Rst_Slave pti e t e'";
     "PB_Slave pti e p t e' p' tp'";
     "IB_PMaster pti e p t e' p' ti'";
     "IBA_PMaster pti e p t e' p'"],
     "(New_State_Is_PA s' e' tp') /\
      (~ELEMENT (FST(L_ad_inE (e' tp'))) (31))
       ==>
      (~P_rqtS (s' ti'))"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC TRANS_TIMES_EQUAL
    THEN IMP_RES_TAC PREC
    THEN REWRITE_ASSUM_TAC ("PStateAbs pti s e p t s' e' p' tp'",[PStateAbs])
    THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
    THEN RES_TAC
    THEN ASM_REWRITE_ASSUM_TAC ("P_rqtS(s' (ti':timeC))",[])
  );;

%-----------------------------------------------------------------------------
   P_RQT_TRUE_ON_TI'_IMP_DELAY_CONDS =
   |- !pti e p t e' p' ti' tp' s s'.
       IBA_PMaster pti e p t e' p' ==>
       IB_PMaster pti e p t e' p' ti' ==>
       PB_Slave pti e p t e' p' tp' ==>
       Rst_Slave pti e t e' ==>
       PStateAbs pti s e p t s' e' p' tp' ==>
       PT_PreC pti s e p t ==>
       PT_Exec pti s e p t ==>
       NTH_TIME_TRUE t(ale_sig_ib p')0 ti' ==>
       tp' > 0 ==>
       NTH_TIME_TRUE t(ale_sig_pb e')0 tp' ==>
       PTAbsSet s e p s' e' p' ==>
       PCSet_Correct s' e' p' ==>
       P_rqtS(s' ti') ==>
       ~(New_State_Is_PA s' e' tp' /\ ~ELEMENT(FST(L_ad_inE(e' tp')))31)
-------------------------------------------------------------------------------%

let P_RQT_TRUE_ON_TI'_IMP_DELAY_CONDS =
   REWRITE_RULE [] (GEN_ALL (DISCH_ALL (CONTRAPOS P_RQT_FALSE_ON_TI')));;

let P_RQT_TRUE_ON_TI' = TAC_PROOF
   (([\"PCSet_Correct s' e' p'\";
      \"PTAbsSet s e p s' e' p'\";
      \"NTH_TIME_TRUE t (ale_sig_pb e') 0 tp'\";
      \"tp' > 0\";
      \"NTH_TIME_TRUE t (ale_sig_ib p') 0 ti'\";
      \"PT_Exec pti s e p t\";
      \"PT_PreC pti s e p t\";
      \"PStateAbs pti s e p t s' e' p' tp'\";
      \"Rst_Slave pti e t e'\";
      \"PB_Slave pti e p t e' p' tp'\";
      \"IB_PMaster pti e p t e' p' ti'\";
      \"IBA_PMaster pti e p t e' p'\"],
      \"~(~ELEMENT (FST(L_ad_inE (e' tp'))) (31) /\\ New_State_Is_PA s' e' tp')
       ==>
      P_rqtS (s' ti')\"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC ALE_SIG_IB_TRUE_AFTER_TP'
    THEN POP_ASSUM_LIST (MAP_EVERY (\\thm. STRIP_ASSUME_TAC thm))
    THEN RES_TAC
    THEN IMP_RES_TAC TI'_AFTER_TP'
    THEN IMP_RES_TAC NEXT_IBUS_TRANS_IS_NTH
    THEN SUBGOAL_THEN \"(ti':timeC) = ti''\" ASSUME_TAC
    THENL [
       % Subgoal 1: (New Subgoal) %
       IMP_RES_TAC TRUE_EVENT_TIMES_EQUAL
    ;
       % Subgoal 2: (Continue) %
       REWRITE_ASSUM_TAC
         (\"~(~ELEMENT(FST(L_ad_inE(e' tp')))31 /\\ New_State_Is_PA s' e' tp')\",
          [DE_MORGAN_THM])
       THEN POP_ASSUM_LIST (MAP_EVERY (\\thm. STRIP_ASSUME_TAC thm))
       THEN RES_TAC
       THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LT_IMP_LE_PRE)
       THEN IMP_RES_TAC STABLE_FALSE_THEN
       THEN ASSUME_TAC (SPECL [\"ti'':timeC\";\"1\"] SUB_LESS_EQ)
       THEN IMP_RES_TAC NEW_P_RQT_TRUE_FROM_TP'_TO_TI'
       THEN IMP_RES_TAC M_LESS_0_LESS
       THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LT_IMP_SUC_LE)
       THEN POP_ASSUM_LIST (MAP_EVERY (\\thm. ASSUME_TAC (REDUCE_RULE thm)))
       THEN ASM_REWRITE_TAC[]
       THEN IMP_RES_TAC (SYM_RULE (SPECL [\"ti'':timeC\";\"1\"] SUB_ADD))
       THEN PURE_ONCE_ASM_REWRITE_TAC[]
       THEN DELETE_ASSUM_TAC \"ti'' = (ti'' - 1) + 1\"
       THEN IMP_RES_TAC P_rqt_ISO
       THEN REWRITE_ASSUM_TAC (\"New_P_Rqt_Is_TRUE s' e'(ti'' - 1)\",
                               [New_P_Rqt_Is_TRUE;New_State_Is_PD])
       THEN ASM_REWRITE_TAC[]
    ]
   );;

%-------------------------------------------------------------------------------
```

```
   P_RQT_FALSE_ON_TI'_IMP_FLOWTHRU_CONDS =
   |- !pti e p t e' p' ti' tp' s s'.
       IBA_PMaster pti e p t e' p' ==>
       IB_PMaster pti e p t e' p' ti' ==>
       PB_Slave pti e p t e' p' tp' ==>
       Rst_Slave pti e t e' ==>
       PStateAbs pti s e p t s' e' p' tp' ==>
       PT_PreC pti s e p t ==>
       PT_Exec pti s e p t ==>
       NTH_TIME_TRUE t(ale_sig_ib p')0 ti' ==>
       tp' > 0 ==> NTH_TIME_TRUE t(ale_sig_pb e')0 tp' ==>
       PTAbsSet s e p s' e' p' ==>
       PCSet_Correct s' e' p' ==>
       ~P_rqtS(s' ti') ==>
       ~ELEMENT(FST(L_ad_inE(e' tp')))31 /\ New_State_Is_PA s' e' tp'
--------------------------------------------------------------------------------%

let P_RQT_FALSE_ON_TI'_IMP_FLOWTHRU_CONDS =
   REWRITE_RULE [] (GEN_ALL (DISCH_ALL (CONTRAPOS P_RQT_TRUE_ON_TI')));;

let MAXWORD = mk_thm
   ([],
    " ! n b. (VAL n b) < (2 EXP (SUC n))"
   );;


%--------------------------------------------------------------------------------

   File:      pt_thms2.ml

   Author:    (c) D.A. Fura 1993

   Date:      7 March 1993

   More theorems used in the P-Port trans-level proof.

--------------------------------------------------------------------------------%

let GREATER_TRANS = TAC_PROOF
   (([],
    "! m n p :num .  m > n ==> n > p ==> m > p"),
    REWRITE_TAC [GREATER]
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC LESS_TRANS
   );;

let PRIOR_FALSE_EVENTS_EXIST = mk_thm
   ([],
    "! (x :time->bool) (m n :num) (t0 t :time) .
     NTH_TIME_FALSE n x t0 t ==>
        (m < n) ==>
          (?t'. t' < t /\ NTH_TIME_FALSE m x t0 t')"
   );;

let GT_IMP_NOT_EQ = mk_thm
   ([], "! (m n :num). m > n ==> ~(m = n)");;

let SUP_INTERVAL_STABLE_FALSE = mk_thm
   ([],
    "! (t0 t9 t1 t2 :time) (x :time->bool) .
     STABLE_FALSE x (t0,t1) ==>
        STABLE_FALSE x (t2,t9) ==>
           (t2 <= t1+1) ==>
              (t0 <= t9) ==>
                 STABLE_FALSE x (t0,t9)"
   );;

let LT_IMP_NOT_EQ = mk_thm
   ([], "! m n :num . (m < n) ==> ~(m = n)");;

let DECN_WORDN_1 = TAC_PROOF
   (([],
     "! (m n :num) .
      (m <= 3) ==>
        (n <= 3) ==>
          (m = n + 1) ==>
            (DECN 1 (WORDN 1 m) = WORDN 1 n)"),
     REWRITE_TAC [DECN]
     THEN REPEAT STRIP_TAC
     THEN COND_CASES_TAC
     THENL [
        % Subgoal 1: "SETN 1 = WORDN 1 n"
                     [ "ZEROS 1(WORDN 1 m)" ] %
        SUBGOAL_THEN "n < 3" ASSUME_TAC
        THENL [
           % Subgoal 1.1: "n < 3" %
           REWRITE_ASSUM_TAC ("n <= 3",[LESS_OR_EQ])
           THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
           THEN ASM_REWRITE_TAC[]
           THEN ASM_REWRITE_ASSUM_TAC ("m = n + 1",[])
           THEN UNDISCH_TAC "m <= 3"
           THEN ASM_REWRITE_TAC[]
           THEN REDUCE_TAC
        ;
           % Subgoal 1.2:  [ "n < 3" ] %
           SUBGOAL_THEN "ZEROS 1(WORDN 1 m) ==> (m=0)" IMP_RES_TAC
           THENL [
             % Subgoal 1.2.1: (New Subgoal) %
             REWRITE_TAC [num_CONV "1";ZEROS;WORDN]
             THEN BETA_TAC
             THEN REDUCE_TAC
             THEN REWRITE_ASSUM_TAC ("m <= 3",[LESS_EQ_3_CASES])
             THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
             THEN ASM_REWRITE_TAC[]
             THEN REDUCE_TAC
           ;
             % Subgoal 1.2.2: (Continue) %
             UNDISCH_TAC "m = n + 1"
             THEN REWRITE_ASSUM_TAC ("n <= 3",[LESS_EQ_3_CASES])
             THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
             THEN ASM_REWRITE_TAC[]
             THEN REDUCE_TAC
           ]
        ]
     ;
        % Subgoal 2: "WORDN 1((VAL 1(WORDN 1 m)) - 1) = WORDN 1 n" %
        IMP_RES_TAC VAL_WORDN_IDENT_1
        THEN ASM_REWRITE_TAC [ADD_SUB]
     ]
    );;

let STABLE_HI_IMP_NOT_STABLE_LO = TAC_PROOF
    (([],
     "! (x :time->wire) (t1 t2 :time) .
      STABLE_HI x (t1,t2) ==> ~STABLE_LO x (t1,t2)"),
     REWRITE_TAC [STABLE_HI;STABLE_LO]
     THEN REPEAT STRIP_TAC
     THEN SPEC_ASSUM_TAC
            ("!t. t1 <= t /\ t <= t2 ==> (x t = HI)","t1:timeC")
     THEN SPEC_ASSUM_TAC
            ("!t. t1 <= t /\ t <= t2 ==> (x t = LO)","t1:timeC")
     THEN ASSUME_TAC (SPEC "t1:timeC" LESS_EQ_REFL)
     THEN RES_TAC
     THEN UNDISCH_TAC "x (t1:timeC) = LO"
     THEN ASM_REWRITE_TAC [prove_constructors_distinct wire]
    );;

let Standard_Assumps = new_definition
    ('Standard_Assumps',
     "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
        (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
        (tp' ti' :timeC) .
      Standard_Assumps pti s e p t s' e' p' tp' ti' =
        PCSet_Correct s' e' p' /\
      PTAbsSet s e p s' e' p' /\
      PT_Exec pti s e p t /\
      PT_PreC pti s e p t /\
      PB_Slave pti e p t e' p' tp' /\
      IB_PMaster pti e p t e' p' ti' /\
      IBA_PMaster pti e p t e' p' /\
      Rst_Slave pti e t e' /\
      PStateAbs pti s e p t s' e' p' tp' /\
      NTH_TIME_TRUE t(ale_sig_pb e')0 tp' /\
      tp' > 0 /\
      NTH_TIME_TRUE t(ale_sig_ib p')0 ti' /\
      ti' > 0"
   );;

let EXPAND_STANDARD_ASSUMPS = TAC_PROOF
   (([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' :timeC) .
     Standard_Assumps pti s e p t s' e' p' tp' ti' ==>
        (PCSet_Correct s' e' p' /\
         PTAbsSet s e p s' e' p' /\
         PT_Exec pti s e p t /\
         PT_PreC pti s e p t /\
         PB_Slave pti e p t e' p' tp' /\
         IB_PMaster pti e p t e' p' ti' /\
         IBA_PMaster pti e p t e' p' /\
         Rst_Slave pti e t e' /\
         PStateAbs pti s e p t s' e' p' tp' /\
         NTH_TIME_TRUE t(ale_sig_pb e')0 tp' /\
         tp' > 0 /\
         NTH_TIME_TRUE t(ale_sig_ib p')0 ti' /\
         ti' > 0)"),
    REWRITE_TAC [Standard_Assumps]
    THEN REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let OFFSET_NEW_P_RQT_TRUE_FROM_TI'_TO_T'SACK = TAC_PROOF
   (([],
    "! (u' :timeC)
       (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' t'sack :timeC) .
       PCSet_Correct s' e' p' ==>
       PT_Exec pti s e p t ==>
       IBA_PMaster pti e p t e' p' ==>
       Rst_Slave pti e t e' ==>
       NTH_TIME_TRUE t(ale_sig_ib p')0 ti' ==>
       (ti' > 0) ==>
       STABLE_FALSE (Sack_Sig_Is_TRUE s' e') (ti',t'sack-1) ==>
         ((ti'+u') < t'sack) ==>
           New_P_Rqt_Is_TRUE s' e' (ti'+u')"),
    INDUCT_TAC
    THEN REWRITE_TAC [ADD_CLAUSES;ADD1;ADD_ASSOC]
    THEN REPEAT STRIP_TAC
    THENL [
       % Subgoal 1: (Base Case) %
       IMP_RES_TAC NTH_TIME_TRUE_X_IMP_X
       THEN IMP_RES_TAC IBUS_ALE_IMP_NEW_P_RQT
    ;
       % Subgoal 2: (Induction Step) %
       ASSUME_TAC (SPECL ["ti'+u'";"1"] LESS_EQ_ADD)
       THEN IMP_RES_TAC LESS_EQ_LESS_TRANS
       THEN RES_TAC
       THEN REWRITE_ASSUM_TAC
              ("STABLE_FALSE(Sack_Sig_Is_TRUE s' e')(ti',t'sack-1)",
               [STABLE_FALSE])
       THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
       THEN SPEC_ASSUM_TAC
              ("!t. ti' <= t /\ t <= (t'sack - 1) ==> ~Sack_Sig_Is_TRUE s' e' t",
               "(ti'+u')+1")
      THEN ASSUME_TAC (SPECL ["ti':timeC";"u':timeC"] LESS_EQ_ADD)
      THEN IMP_RES_TAC LESS_EQ_TRANS
      THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LT_IMP_LE_PRE)
      THEN RES_TAC
      THEN SUBGOAL_THEN "P_rqtS (s' ((ti'+u')+1))" ASSUME_TAC
      THENL [
         % Subgoal 2.1: (New Subgoal) %
         IMP_RES_TAC P_rqt_ISO
         THEN REWRITE_ASSUM_TAC
                ("New_P_Rqt_Is_TRUE s' e'(ti' + u')",
                 [New_P_Rqt_Is_TRUE;New_State_Is_PD])
         THEN ASM_REWRITE_TAC[]
      ;
         % Subgoal 2.2: (Continue) %
         IMP_RES_TAC RST_FALSE
         THEN NRULE_ASSUM_TAC
                ("~Sack_Sig_Is_TRUE s' e'((ti' + u') + 1)",
                 (BETA_RULE o (REWRITE_RULE[Sack_Sig_Is_TRUE;New_State_Is_PD])))
         THEN IMP_RES_TAC P_rqt_ISO
         THEN ASM_REWRITE_TAC [New_P_Rqt_Is_TRUE;New_State_Is_PD;COND_TRUE_TRUE;
                               COND_TRUE_CHOICES]
      ]
   ]
  );;

let NEW_P_RQT_TRUE_FROM_TI'_TO_T'SACK = TAC_PROOF
  (([],
   "! (t' :timeC)
      (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
      (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
      (tp' ti' t'sack :timeC) .
      PCSet_Correct s' e' p' ==>
       PT_Exec pti s e p t ==>
        IBA_PMaster pti e p t e' p' ==>
         Rst_Slave pti e t e' ==>
          NTH_TIME_TRUE t(ale_sig_ib p')0 ti' ==>
           (ti' > 0) ==>
            STABLE_FALSE (Sack_Sig_Is_TRUE s' e') (ti',t'sack-1) ==>
             (ti' <= t') ==>
              (t' < t'sack) ==>
               New_P_Rqt_Is_TRUE s' e' t'"),
   REPEAT STRIP_TAC
   THEN IMP_RES_TAC (SPEC "t'-ti'" OFFSET_NEW_P_RQT_TRUE_FROM_TI'_TO_T'SACK)
   THEN SPECL_ASSUM_TAC
          ("!t' ti'''. (ti' + (t' - ti''')) < t'sack ==>
             New_P_Rqt_Is_TRUE s' e'(ti' + (t' - ti'''))",
           ["t':timeC";"ti':timeC"])
   THEN IMP_RES_TAC
          (SPECL ["t':timeC";"ti':timeC"]
                 (PURE_ONCE_REWRITE_RULE [ADD_SYM] SUB_ADD))
   THEN ASM_REWRITE_ASSUM_TAC
          ("(ti' + (t' - ti')) < t'sack ==>
            New_P_Rqt_Is_TRUE s' e'(ti' + (t' - ti'))",[])
   THEN ASM_REWRITE_TAC[]
  );;

let SACK_SIG_FALSE_DURING_DATA_0 = TAC_PROOF
  (([],
   "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
      (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
      (ti' t'sack t'rdy0 :timeC) .
      PCSet_Correct s' e' p' ==>
       PT_Exec pti s e p t ==>
        IBA_PMaster pti e p t e' p' ==>
         NTH_TIME_TRUE t(ale_sig_ib p')0 ti' ==>
          (ti' > 0) ==>
           STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (ti'+1,t'rdy0) ==>
            STABLE_FALSE (Sack_Sig_Is_TRUE s' e') (ti',t'rdy0-1)"),
   REWRITE_TAC [STABLE_TRUE_THEN_FALSE;bsig;BSel;STABLE_FALSE;Sack_Sig_Is_TRUE]
   THEN BETA_TAC
   THEN REPEAT STRIP_TAC
   THENL [
      % Subgoal 1: "ti' <= (t'rdy0 - 1)" %
      IMP_RES_TAC (SPECL ["ti'+1";"t'rdy0:timeC";"1"] LESS_EQ_MONO_SUB)
      THEN RULE_ASSUM_TAC (REWRITE_RULE [SPECL ["ti':timeC";"1"] ADD_SUB])
      THEN ASM_REWRITE_TAC[]
   ;
      % Subgoal 2: [ "~SND(I_srdy_E(e' t'))" ] %
      SPEC_ASSUM_TAC
        ("!t. (ti' + 1) <= t /\ t < t'rdy0 ==> SND(I_srdy_E(e' t))","t':timeC")
      THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
      THEN ASSUME_TAC (SPECL ["ti':timeC";"1"] LESS_EQ_ADD)
      THEN IMP_RES_TAC LESS_EQ_TRANS
      THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LE_PRE_IMP_LT)
      THEN ASM_CASES_TAC "(ti'+1)<=t'"
      THENL [
         % Subgoal 2.1: [ "(ti' + 1) <= t'" ] %
         RES_TAC
         THEN RES_TAC
      ;
         % Subgoal 2.2:  [ "~(ti' + 1) <= t'" ] %
         IMP_RES_TAC NOT_LESS_EQ_LESS
         THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LT_SUC_IMP_LE)
         THEN IMP_RES_TAC LESS_EQUAL_ANTISYM
         THEN DELETE_ASSUM_TAC "t' = (ti':timeC)"
         THEN ASM_REWRITE_ASSUM_TAC ("NTH_TIME_TRUE t(ale_sig_ib p')0 ti'",[])
         THEN IMP_RES_TAC NTH_TIME_TRUE_X_IMP_X
         THEN IMP_RES_TAC IBUS_ALE_IMP_PA
         THEN UNDISCH_TAC "New_State_Is_PD s' e' t'"
         THEN REWRITE_ASSUM_TAC ("New_State_Is_PA s' e' t'",[New_State_Is_PA])
         THEN IMP_RES_TAC PA_IMP_NOT_PD
         THEN ASM_REWRITE_TAC [New_State_Is_PD]
      ]
   ]
  );;

let P_LOAD_TRUE_THEN_STABLE_FALSE_FROM_TP'_TO_T'SACK = TAC_PROOF
   (([],
    "! (t' :timeC)
       (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' t'sack :timeC) .
     (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
      STABLE_FALSE (Sack_Sig_Is_TRUE s' e') (ti',t'sack-1)) ==>
       TRUE_THEN_STABLE_FALSE (\u'. P_loadS (s' u')) (tp',t'sack)"),
    REWRITE_TAC [Standard_Assumps;TRUE_THEN_STABLE_FALSE]
    THEN BETA_TAC
    THEN REPEAT STRIP_TAC
    THENL [
      % Subgoal 1: "tp' <= t'sack" %
      REWRITE_ASSUM_TAC
        ("STABLE_FALSE(Sack_Sig_Is_TRUE s' e')(ti',t'sack - 1)",[STABLE_FALSE])
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN ASSUME_TAC (SPECL ["t'sack:timeC";"1"] SUB_LESS_EQ)
      THEN IMP_RES_TAC NTH_TRANS_CAUSAL
      THEN IMP_RES_TAC LESS_EQ_TRANS
    ;
      % Subgoal 2: "P_loadS(s' tp')" %
      SUBGOAL_THEN "~P_rqtS(s' (tp':timeC))" ASSUME_TAC
      THENL [
         % Subgoal 2.1: (New Subgoal) %
         PREC_TAC
      ;
         % Subgoal 2.2: (Continue) %
         UNDISCH_TAC "~P_rqtS(s' (tp':timeC))"
         THEN SUBGOAL_THEN "tp' = (tp'-1)+1" (\thm. PURE_ONCE_REWRITE_TAC [thm])
         THENL [
           % Subgoal 2.2.1: (New Subgoal) %
           IMP_RES_TAC (RIMP ONE_LESS_EQ)
           THEN IMP_RES_TAC
                  (SPECL ["tp':timeC";"1"] (SYM_RULE SUB_ADD))
         ;
           % Subgoal 2.2.2: (Continue) %
           IMP_RES_TAC P_load_ISO
        THEN IMP_RES_TAC P_rqt_ISO
        THEN ASM_REWRITE_TAC[]
        THEN DISCH_TAC
        THEN ASM_REWRITE_TAC[]
      ]
    ]
;
  % Subgoal 3: [ "tp' < t'" ]
              [ "t' <= t'sack" ]
              [ "P_loadS(s' t')" ] %
  UNDISCH_TAC "P_loadS(s' (t':timeC))"
  THEN SUBGOAL_THEN "t' = (t'-1)+1" (\thm. PURE_ONCE_REWRITE_TAC [thm])
  THENL [
     % Subgoal 3.1: (New Subgoal) %
     IMP_RES_TAC (RIMP ONE_LESS_EQ)
     THEN IMP_RES_TAC LESS_EQ_LESS_TRANS
     THEN IMP_RES_TAC LT_IMP_LE
     THEN IMP_RES_TAC (SYM_RULE (SPECL ["t':timeC";"1"] SUB_ADD))
  ;
     % Subgoal 3.2: (Continue) %
     IMP_RES_TAC IBUS_TRANS_EXISTS
     THEN IMP_RES_TAC NEXT_IBUS_TRANS_IS_NTH
     THEN SUBGOAL_THEN
            "ti'' = (ti':timeC)" (\thm. RULE_ASSUM_TAC (REWRITE_RULE [thm]))
     THENL [
       % Subgoal 3.2.1: (New Subgoal) %
       IMP_RES_TAC TRUE_EVENT_TIMES_EQUAL
     ;
       % Subgoal 3.2.2: (Continue) %
       REWRITE_ASSUM_TAC
         ("STABLE_FALSE_THEN_TRUE(ale_sig_ib p')(tp',ti')",
          [STABLE_FALSE_THEN_TRUE])
       THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
     ]
     THEN IMP_RES_TAC NEW_P_RQT_TRUE_FROM_TI'_TO_T'SACK
     THEN IMP_RES_TAC NEW_P_RQT_TRUE_FROM_TP'_TO_TI'
     THEN NRULE_ASSUM_TAC
            ("!ti'. STABLE_FALSE(ale_sig_ib p')(tp',ti' - 1) ==>
              (!t'. tp' <= t' ==> t' <= ti' ==> New_P_Rqt_Is_TRUE s' e' t')",
             ((REWRITE_RULE [STABLE_FALSE]) o (SPEC "ti':timeC")))
     THEN ASM_CASES_TAC "tp' <= (ti' - 1)"
     THENL [
       % Subgoal 3.2.2.1: [ "tp' <= (ti' - 1)" ] %
       SUBGOAL_THEN
         "(!t. tp' <= t /\ t <= (ti' - 1) ==> ~ale_sig_ib p' t)" ASSUME_TAC
       THENL [
         % Subgoal 3.2.2.1.1: (New Subgoal) %
         REPEAT STRIP_TAC
         THEN SPEC_ASSUM_TAC
                ("!t. tp' <= t /\ t < ti' ==> ~ale_sig_ib p' t","t'':timeC")
         THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
         THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LE_PRE_IMP_LT)
         THEN RES_TAC
       ;
         % Subgoal 3.2.2.1.2: (Continue) %
         ASM_REWRITE_ASSUM_TAC
           ("tp' <= (ti' - 1) /\
             (!t. tp' <= t /\ t <= (ti' - 1) ==> ~ale_sig_ib p' t) ==>
             (!t'. tp' <= t' ==> t' <= ti' ==> New_P_Rqt_Is_TRUE s' e' t')",
            [])
         THEN SPEC_ASSUM_TAC
               ("!t'. ti'<=t' ==> t'<t'sack ==> New_P_Rqt_Is_TRUE s' e' t'",
                "t'-1")
         THEN SPEC_ASSUM_TAC
               ("!t'. tp'<=t' ==> t'<=ti' ==> New_P_Rqt_Is_TRUE s' e' t'",
                "t'-1")
         THEN ASM_CASES_TAC "ti' <= (t' - 1)"
         THENL [
           % Subgoal 3.2.2.1.2.1:  [ "ti' <= (t' - 1)" ] %
           IMP_RES_TAC (RIMP ONE_LESS_EQ)
           THEN IMP_RES_TAC LESS_EQ_LESS_TRANS
           THEN IMP_RES_TAC LESS_LESS_EQ_TRANS
```

```
                THEN IMP_RES_TAC LT_IMP_LE
                THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LE_IMP_PRE_LT)
                THEN RES_TAC
              ;
                % Subgoal 3.2.2.1.2.2: [ "~ti' <= (t' - 1)" ] %
                IMP_RES_TAC NOT_LESS_EQ_LESS
                THEN IMP_RES_TAC LT_IMP_LE
                THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LT_IMP_LE_PRE)
                THEN RES_TAC
              ]
              THEN REWRITE_ASSUM_TAC
                     ("New_P_Rqt_Is_TRUE s' e'(t' - 1)",
                      [New_P_Rqt_Is_TRUE;New_State_Is_PD])
              THEN IMP_RES_TAC P_load_ISO
              THEN ASM_REWRITE_TAC[]
            ]
          ;
            % Subgoal 3.2.2.2: [ "~tp' <= (ti' - 1)" ] %
            IMP_RES_TAC NOT_LESS_EQ_LESS
            THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
            THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] PRE_LT_IMP_LE)
            THEN SUBGOAL_THEN
                   "ti' = (tp':timeC)" (\thm. RULE_ASSUM_TAC (REWRITE_RULE [thm]))
            THENL [
              % Subgoal 3.2.2.2.1: (New Subgoal) %
              IMP_RES_TAC LESS_EQUAL_ANTISYM
            ;
              % Subgoal 3.2.2.2.2: (Continue) %
              SPEC_ASSUM_TAC
                ("!t'. tp' <= t' ==> t' < t'sack ==> New_P_Rqt_Is_TRUE s' e' t'",
                 "t'-1")
              THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LT_IMP_LE_PRE)
              THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
              THEN IMP_RES_TAC LESS_EQ_LESS_TRANS
              THEN IMP_RES_TAC LT_IMP_LE
              THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LE_IMP_PRE_LT)
              THEN RES_TAC
              THEN REWRITE_ASSUM_TAC
                     ("New_P_Rqt_Is_TRUE s' e'(t' - 1)",
                      [New_P_Rqt_Is_TRUE;New_State_Is_PD])
              THEN IMP_RES_TAC P_load_ISO
              THEN ASM_REWRITE_TAC[]
            ]
          ]
        ]
      ]
    );;

let P_DOWN_STABLE_FALSE_THEN_TRUE_FROM_TP'_TO_T'RDY0 = TAC_PROOF
    (([],
     "! (t' :timeC)
        (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
        (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
        (tp' ti' t'sack :timeC) .
      (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
       STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (ti'+1,t'rdy0)) ==>
        STABLE_FALSE_THEN_TRUE (\u'. P_downS (s' u')) (tp'+1,t'rdy0+1)"),
    REWRITE_TAC [Standard_Assumps;STABLE_FALSE_THEN_TRUE]
    THEN BETA_TAC
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC SACK_SIG_FALSE_DURING_DATA_0
    THENL [
       % Subgoal 1: "(tp' + 1) <= (t'rdy0 + 1)" %
       REWRITE_ASSUM_TAC
         ("STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(ti' + 1,t'rdy0)",
          [STABLE_TRUE_THEN_FALSE])
       THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
       THEN ASSUME_TAC (SPECL ["t'rdy0:timeC";"1"] LESS_EQ_ADD)
       THEN IMP_RES_TAC NTH_TRANS_CAUSAL
       THEN IMP_RES_TAC
              (RIMP (SPECL ["tp':timeC";"ti':timeC";"1"] LESS_EQ_MONO_ADD_EQ))
       THEN IMP_RES_TAC LESS_EQ_TRANS
```

```
;
 % Subgoal 2:  [ "(tp' + 1) <= t'" ]
              [ "t' < (t'rdy0 + 1)" ]
              [ "P_downS(s' t')" ] %
 NRULE_ASSUM_TAC
   ("STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(ti' + 1,t'rdy0)",
    (BETA_RULE o (REWRITE_RULE [STABLE_TRUE_THEN_FALSE;bsig;BSel])))
 THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
 THEN SPEC_ASSUM_TAC
       ("!t. (ti' + 1) <= t /\ t < t'rdy0 ==> SND(I_srdy_E(e' t))", "t'-1")
 THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LT_SUC_IMP_LE)
 THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
 THEN ASSUME_TAC (SPECL ["tp':timeC";"1"] LESS_EQ_ADD)
 THEN IMP_RES_TAC LESS_EQ_TRANS
 THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LE_IMP_PRE_LT)
 THEN ASM_CASES_TAC "(ti' + 1) <= (t' - 1)"
 THENL [
    % Subgoal 2.1: [ "(ti' + 1) <= (t' - 1)" ] %
    RES_TAC
    THEN UNDISCH_TAC "P_downS(s' (t':timeC))"
    THEN SUBGOAL_THEN "t' = (t'-1)+1" (\thm. PURE_ONCE_REWRITE_TAC [thm])
    THENL [
       % Subgoal 2.1.1: (New Subgoal) %
       IMP_RES_TAC (RIMP ONE_LESS_EQ)
       THEN IMP_RES_TAC LESS_EQ_LESS_TRANS
       THEN IMP_RES_TAC LT_IMP_LE
       THEN IMP_RES_TAC (SYM_RULE (SPECL ["t':timeC";"1"] SUB_ADD))
    ;
       % Subgoal 2.1.2: (Continue) %
       IMP_RES_TAC
        (SPECL ["s':timeC->pc_state";"e':timeC->pc_env";"p':timeC->pc_out";
                "t'-1"] (GEN_ALL P_down_ISO))
       THEN ASM_REWRITE_TAC[]
    ]
 ;
    % Subgoal 2.2:  [ "~(ti' + 1) <= (t' - 1)" ] %
    IMP_RES_TAC NOT_LESS_EQ_LESS
    THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LT_SUC_IMP_LE)
    THEN IMP_RES_TAC (SPECL ["ti'+1";"t':timeC";"1"] LESS_EQ_MONO_SUB)
    THEN SUBGOAL_THEN
           "(tp'+1)-1=tp'" (\thm. RULE_ASSUM_TAC (REWRITE_RULE [thm]))
    THENL [
      % Subgoal 2.2.1: (New Subgoal) %
      REWRITE_TAC [ADD_SUB]
    ;
      % Subgoal 2.2.2: (Continue) %
      UNDISCH_TAC "P_downS(s' (t':timeC))"
      THEN SUBGOAL_THEN
        "t' = (t'-1)+1" (\thm. PURE_ONCE_REWRITE_TAC [thm])
      THENL [
         % Subgoal 2.2.2.1: (New Subgoal) %
         IMP_RES_TAC (RIMP ONE_LESS_EQ)
         THEN IMP_RES_TAC LESS_EQ_LESS_TRANS
         THEN IMP_RES_TAC LT_IMP_LE
         THEN IMP_RES_TAC (SYM_RULE (SPECL ["t':timeC";"1"] SUB_ADD))
      ;
         % Subgoal 2.2.2.2: (Continue) %
         IMP_RES_TAC IBUS_TRANS_EXISTS
         THEN IMP_RES_TAC NEXT_IBUS_TRANS_IS_NTH
         THEN SUBGOAL_THEN
                "ti'' = (ti':timeC)"
                (\thm. RULE_ASSUM_TAC (REWRITE_RULE [thm]))
         THENL [
           % Subgoal 2.2.2.2.1: (New Subgoal) %
           IMP_RES_TAC TRUE_EVENT_TIMES_EQUAL
         ;
           % Subgoal 2.2.2.2.2: (Continue) %
           REWRITE_ASSUM_TAC
             ("STABLE_FALSE_THEN_TRUE(ale_sig_ib p')(tp',ti')",
              [STABLE_FALSE_THEN_TRUE])
           THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
           THEN IMP_RES_TAC NEW_STATE_PD_FALSE_FROM_TP'_TO_TI'
THEN NRULE_ASSUM_TAC
       ("!ti'. STABLE_FALSE(ale_sig_ib p')(tp',ti' - 1) ==>
               (!t'. tp' <= t' ==> t' <= ti' ==>
               ~New_State_Is_PD s' e' t')",
        ((REWRITE_RULE [STABLE_FALSE]) o (SPEC "ti':timeC")))

THEN ASM_CASES_TAC "tp' <= (ti' - 1)"
THENL [
  % Subgoal 2.2.2.2.2.1: [ "tp' <= (ti' - 1)" ] %
  SUBGOAL_THEN
    "(!t. tp' <= t /\ t <= (ti' - 1) ==>
       ~ale_sig_ib p' t)" ASSUME_TAC
  THENL [
    % Subgoal 2.2.2.2.2.1.1: (New Subgoal) %
    REPEAT STRIP_TAC
    THEN SPEC_ASSUM_TAC
          ("!t. tp' <= t /\ t < ti' ==>
             ~ale_sig_ib p' t","t'':timeC")
    THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
    THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LE_PRE_IMP_LT)
    THEN RES_TAC
  ;
    % Subgoal 2.2.2.2.2.1.2: (Continue) %
    ASM_REWRITE_ASSUM_TAC
      ("tp' <= (ti' - 1) /\
        (!t. tp' <= t /\ t <= (ti' - 1) ==>
         ~ale_sig_ib p' t) ==> (!t'. tp' <= t' ==> t' <= ti'
         ==> ~New_State_Is_PD s' e' t')",[])
    THEN SPEC_ASSUM_TAC
          ("!t'. tp' <= t' ==> t' <= ti' ==>
                 ~New_State_Is_PD s' e' t'","t'-1")
    THEN RES_TAC
    THEN REWRITE_ASSUM_TAC
          ("~New_State_Is_PD s' e'(t' - 1)",[New_State_Is_PD])
    THEN IMP_RES_TAC
          (SPECL ["s':timeC->pc_state";"e':timeC->pc_env";
                  "p':timeC->pc_out";"t'-1"]
                 (GEN_ALL P_down_ISO))
    THEN ASM_REWRITE_TAC[]
  ]
;
  % Subgoal 2.2.2.2.2.2: [ "~tp' <= (ti' - 1)" ] %
  IMP_RES_TAC NOT_LESS_EQ_LESS
  THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] PRE_LT_IMP_LE)
  THEN SUBGOAL_THEN
         "tp' = (ti':timeC)"
         (\thm. RULE_ASSUM_TAC (REWRITE_RULE [thm]))
  THENL [
    % Subgoal 2.2.2.2.2.2.1: (New Subgoal) %
    IMP_RES_TAC LESS_EQUAL_ANTISYM
  ;
    % Subgoal 2.2.2.2.2.2.2: (Continue) %
    SUBGOAL_THEN "(t'-1) = ti'" (\thm. REWRITE_TAC [thm])
    THENL [
      % Subgoal 2.2.2.2.2.2.2.1: (New Subgoal) %
      IMP_RES_TAC LESS_EQUAL_ANTISYM
    ;
      % Subgoal 2.2.2.2.2.2.2: (Continue) %
      IMP_RES_TAC IBUS_ALE_IMP_PA
      THEN REWRITE_ASSUM_TAC
            ("New_State_Is_PA s' e' ti'",[New_State_Is_PA])
      THEN ASSUME_TAC
            (SPECL ["s':timeC->pc_state";"e':timeC->pc_env";
                    "p':timeC->pc_out";"ti':timeC"]
                   (GEN_ALL P_down_ISO))
      THEN RES_TAC
      THEN ASM_REWRITE_TAC
            [prove_constructors_distinct pfsm_ty_Axiom]
    ]
  ]
 ]
]
```

```
                ]
             ]
          ]
       ;
          % Subgoal 3: "P_downS(s'(t'rdy0 + 1))" %
          IMP_RES_TAC NTH_TIME_TRUE_X_IMP_X
          THEN IMP_RES_TAC NEW_STATE_PD_FROM_TI'_TO_T'SACK_1
          THEN NRULE_ASSUM_TAC
                 ("STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(ti' + 1,t'rdy0)",
                  (BETA_RULE o (REWRITE_RULE [STABLE_TRUE_THEN_FALSE;bsig;BSel])))
          THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
          THEN SPEC_ASSUM_TAC
                 ("!t'. (ti' + 1) <= t' ==> t' <= t'rdy0 ==>
                         New_State_Is_PD s' e' t'","t'rdy0:timeC")
          THEN ASSUME_TAC (SPEC "t'rdy0:timeC" LESS_EQ_REFL)
          THEN RES_TAC
          THEN REWRITE_ASSUM_TAC ("New_State_Is_PD s' e' t'rdy0",[New_State_Is_PD])
          THEN IMP_RES_TAC P_down_ISO
          THEN ASM_REWRITE_TAC[]
       ]
      );;

let OFFSET_P_SIZE_STABLE_FROM_TP'_TO_T'RDY0 = TAC_PROOF
   (([],
    "! (u' :timeC)
       (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' t'rdy0 :timeC) .
     (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
      STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (ti'+1,t'rdy0) /\
      ((tp'+u'+1) <= t'rdy0))
      ==>
     (P_sizeS (s' (tp'+u'+1)) = SUBARRAY (SND(L_ad_inE (e' tp'))) (1,0))"),
    INDUCT_TAC
    THEN REPEAT STRIP_TAC
    THEN RULE_ASSUM_TAC (REWRITE_RULE [ADD1;ADD_CLAUSES;ADD_ASSOC])
    THENL [
          % Subgoal 1: (Base Case) %
          IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
          THEN IMP_RES_TAC SACK_SIG_FALSE_DURING_DATA_0
          THEN IMP_RES_TAC P_LOAD_TRUE_THEN_STABLE_FALSE_FROM_TP'_TO_T'SACK
          THEN NRULE_ASSUM_TAC
                 ("TRUE_THEN_STABLE_FALSE(\u'. P_loadS(s' u'))(tp',t'rdy0)",
                  (BETA_RULE o (REWRITE_RULE [TRUE_THEN_STABLE_FALSE])))
          THEN IMP_RES_TAC P_size_ISO
          THEN ASM_REWRITE_TAC [ADD_CLAUSES]
    ;
          % Subgoal 2: (Induction Step) %
          ASSUME_TAC (SPECL ["(tp'+u')+1";"1"] LESS_EQ_ADD)
          THEN IMP_RES_TAC LESS_EQ_TRANS
          THEN RES_TAC
          THEN POP_ASSUM (\thm. ALL_TAC)
          THEN POP_ASSUM (\thm. ALL_TAC)
          THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
          THEN IMP_RES_TAC SACK_SIG_FALSE_DURING_DATA_0
          THEN IMP_RES_TAC P_LOAD_TRUE_THEN_STABLE_FALSE_FROM_TP'_TO_T'SACK
          THEN NRULE_ASSUM_TAC
                 ("TRUE_THEN_STABLE_FALSE(\u'. P_loadS(s' u'))(tp',t'rdy0)",
                  (BETA_RULE o (REWRITE_RULE [TRUE_THEN_STABLE_FALSE])))
          THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
          THEN SPEC_ASSUM_TAC
                 ("!t. tp' < t /\ t <= t'rdy0 ==> ~P_loadS(s' t)","(tp'+u')+1")
          THEN ASSUME_TAC (SPECL ["tp':timeC";"u':timeC"] LESS_EQ_ADD)
          THEN ASSUME_TAC (REWRITE_RULE [ADD1] (SPEC "tp'+u'" LESS_SUC_REFL))
          THEN IMP_RES_TAC LESS_EQ_LESS_TRANS
          THEN IMP_RES_TAC P_DOWN_STABLE_FALSE_THEN_TRUE_FROM_TP'_TO_T'RDY0
          THEN NRULE_ASSUM_TAC
                 ("STABLE_FALSE_THEN_TRUE(\u'. P_downS(s' u'))(tp' + 1,t'rdy0 + 1)",
                  (BETA_RULE o (REWRITE_RULE [STABLE_FALSE_THEN_TRUE])))
          THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
          THEN SPEC_ASSUM_TAC
                 ("!t. (tp' + 1) <= t /\ t < (t'rdy0 + 1) ==> ~P_downS(s' t)",
            "(tp'+u')+1")
        THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LE_IMP_LT_SUC)
        THEN IMP_RES_TAC
              (SPECL ["tp':timeC";"tp'+u'";"1"] (RIMP LESS_EQ_MONO_ADD_EQ))
        THEN RES_TAC
        THEN IMP_RES_TAC P_size_ISO
        THEN ASM_REWRITE_TAC [ADD_CLAUSES;ADD1;ADD_ASSOC]
    ]
   );;

let P_SIZE_STABLE_FROM_TP'_TO_T'RDY0 = TAC_PROOF
   (([],
    "! (t' :timeC)
       (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' t'rdy0 :timeC) .
     (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
      STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (ti'+1,t'rdy0) /\
      ((tp'+1) <= t') /\
      (t' <= t'rdy0))
      ==>
     (P_sizeS (s' t') = SUBARRAY (SND(L_ad_inE (e' tp'))) (1,0))"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC (SPEC "t'-tp'+1" OFFSET_P_SIZE_STABLE_FROM_TP'_TO_T'RDY0)
    THEN POP_ASSUM (\thm. ALL_TAC)
    THEN POP_ASSUM (\thm. ALL_TAC)
    THEN SPECL_ASSUM_TAC
           ("!t' tp'''. (tp' + ((t' - (tp''' + 1)) + 1)) <= t'rdy0 ==>
              (P_sizeS(s'(tp' + ((t' - (tp''' + 1)) + 1))) =
                SUBARRAY(SND(L_ad_inE(e' tp')))(1,0))",
            ["t':timeC";"tp':timeC"])
    THEN SUBGOAL_THEN
            "(tp' + ((t' - (tp' + 1)) + 1)) = t'"
            (\thm. RULE_ASSUM_TAC (REWRITE_RULE [thm]))
    THENL [
       % Subgoal 1: -New subgoal- "tp' + ((t' - (tp' + 1)) + 1) = t'" %
       REWRITE_TAC [SYM_RULE (ASSOC_SUB_SUB1)]
       THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "1<=1"))
       THEN IMP_RES_TAC SUC_LE_IMP_LE
       THEN SUBGOAL_THEN "1 <= (t' - tp')" ASSUME_TAC
       THENL [
          % Subgoal 1.1: "1 <= (t' - tp')"
                              [ "(tp' + 1) <= t'" ] %
          REWRITE_TAC
            [SYM_RULE (SPECL ["1";"t'-tp'";"tp':timeC"] LESS_EQ_MONO_ADD_EQ)]
          THEN ASSUME_TAC (SPEC "tp':timeC" LESS_EQ_REFL)
          THEN IMP_RES_TAC
                 (SPECL ["t':timeC";"tp':timeC";"tp':timeC"] ASSOC_SUB_ADD1)                    THEN
ASM_REWRITE_TAC [SUB_EQUAL_0;ADD_CLAUSES]
          THEN PURE_ONCE_REWRITE_TAC [ADD_SYM]
          THEN ASM_REWRITE_TAC[]
        ;
          % Subgoal 1.2: [ "1 <= (t' - tp')" ] %
          IMP_RES_TAC (SPECL ["t'-tp'";"1";"1"] ASSOC_SUB_ADD1)
          THEN ASM_REWRITE_TAC[SUB_EQUAL_0;ADD_CLAUSES]
          THEN PURE_ONCE_REWRITE_TAC [ADD_SYM]
          THEN ASSUME_TAC (SPEC "tp':timeC" LESS_EQ_REFL)
          THEN IMP_RES_TAC
                 (SPECL ["t':timeC";"tp':timeC";"tp':timeC"] ASSOC_SUB_ADD1)
          THEN ASM_REWRITE_TAC [SUB_EQUAL_0;ADD_CLAUSES]
       ]
     ;
       % Subgoal 2 %
       RES_TAC
    ]
   );;

% This should have been proven above. %
let P_SIZE_STABLE_AT_T'RDY0_PLUS_1 = TAC_PROOF
   (([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
```

```
        (tp' ti' t'rdy0 :timeC) .
      (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
       STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (ti'+1,t'rdy0))
       ==>
      (P_sizeS (s' (t'rdy0 + 1)) = SUBARRAY (SND(L_ad_inE (e' tp'))) (1,0))"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
    THEN IMP_RES_TAC P_size_ISO
    THEN ASM_REWRITE_TAC[]
    THEN IMP_RES_TAC SACK_SIG_FALSE_DURING_DATA_0
    THEN IMP_RES_TAC P_LOAD_TRUE_THEN_STABLE_FALSE_FROM_TP'_TO_T'SACK
    THEN NRULE_ASSUM_TAC
           ("TRUE_THEN_STABLE_FALSE(\u'. P_loadS(s' u'))(tp',t'rdy0)",
            (BETA_RULE o (REWRITE_RULE [TRUE_THEN_STABLE_FALSE])))
    THEN IMP_RES_TAC P_DOWN_STABLE_FALSE_THEN_TRUE_FROM_TP'_TO_T'RDY0
    THEN NRULE_ASSUM_TAC
           ("STABLE_FALSE_THEN_TRUE(\u'. P_downS(s' u'))(tp' + 1,t'rdy0 + 1)",
            (BETA_RULE o (REWRITE_RULE [STABLE_FALSE_THEN_TRUE])))
    THEN IMP_RES_TAC P_SIZE_STABLE_FROM_TP'_TO_T'RDY0
    THEN SPEC_ASSUM_TAC
           ("!t'. (tp' + 1) <= t' ==> t' <= t'rdy0 ==>
             (P_sizeS(s' t') = SUBARRAY(SND(L_ad_inE(e' tp')))(1,0))",
            "t'rdy0:timeC")
    THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
    THEN SPEC_ASSUM_TAC
           ("!t. (tp' + 1) <= t /\ t < (t'rdy0 + 1) ==> ~P_downS(s' t)",
            "t'rdy0:timeC")
    THEN SPEC_ASSUM_TAC
           ("!t. tp' < t /\ t <= t'rdy0 ==> ~P_loadS(s' t)","t'rdy0:timeC")
    THEN ASSUME_TAC (SPEC "t'rdy0:timeC" LESS_EQ_REFL)
    THEN ASSUME_TAC (SPEC "t'rdy0:timeC" (REWRITE_RULE [ADD1] LESS_SUC_REFL))
    THEN IMP_RES_TAC NTH_TRANS_CAUSAL
    THEN ASSUME_TAC (SPEC "ti':timeC" (REWRITE_RULE [ADD1] LESS_SUC_REFL))
    THEN REWRITE_ASSUM_TAC
           ("STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(ti' + 1,t'rdy0)",
            [STABLE_TRUE_THEN_FALSE])
    THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
    THEN IMP_RES_TAC LESS_EQ_LESS_TRANS
    THEN IMP_RES_TAC LESS_LESS_EQ_TRANS
    THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LT_IMP_SUC_LE)
    THEN RES_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let I_LAST_FOR_BLOCK_SIZE_0 = TAC_PROOF
   (([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' t'rdy0 :timeC) .
     (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
      (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0) = WORDN 1 0) /\
      STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (ti'+1,t'rdy0)) ==>
         STABLE_LO (bsig I_last_O p') (ti'+1,t'rdy0)"),
    REWRITE_TAC [bsig;BSel;STABLE_LO]
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC (REWRITE_RULE [bsig;BSel] IB_READY_ASSUMPS)
    THENL [
       % Subgoal 1: "(ti' + 1) <= t'rdy0" %
       REWRITE_ASSUM_TAC
          ("STABLE_TRUE_THEN_FALSE(\t. SND(I_srdy_E(e' t)))(ti' + 1,t'rdy0)",
           [STABLE_TRUE_THEN_FALSE])
       THEN ASM_REWRITE_TAC[]
    ;
       % Subgoal 2:
         "(\t. SND(I_last_O(p' t)))t' = LO"
         [ "STABLE_TRUE_THEN_FALSE(\t. SND(I_srdy_E(e' t)))(ti' + 1,t'rdy0)" ]
         [ "(ti' + 1) <= t'" ]
         [ "t' <= t'rdy0" ] %
       BETA_TAC
       THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
       THEN IMP_RES_TAC (REWRITE_RULE [bsig;BSel] SACK_SIG_FALSE_DURING_DATA_0)
       THEN IMP_RES_TAC
```

```
            (REWRITE_RULE [bsig;BSel] P_SIZE_STABLE_FROM_TP'_TO_T'RDY0)
     THEN IMP_RES_TAC NTH_TRANS_CAUSAL
     THEN IMP_RES_TAC
            (SPECL ["tp':timeC";"ti':timeC";"1"] (RIMP LESS_EQ_MONO_ADD_EQ))
     THEN IMP_RES_TAC LESS_EQ_TRANS
     THEN IMP_RES_TAC
            (REWRITE_RULE
               [bsig;BSel] P_DOWN_STABLE_FALSE_THEN_TRUE_FROM_TP'_TO_T'RDY0)
     THEN NRULE_ASSUM_TAC
            ("STABLE_FALSE_THEN_TRUE(\u'. P_downS(s' u'))(tp' + 1,t'rdy0 + 1)",
             (BETA_RULE o (REWRITE_RULE [STABLE_FALSE_THEN_TRUE])))
     THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
     THEN SPEC_ASSUM_TAC
            ("!t. (tp' + 1) <= t /\ t < (t'rdy0 + 1) ==> ~P_downS(s' t)",
             "t':timeC")
     THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LE_IMP_LT_SUC)
     THEN IMP_RES_TAC NTH_TIME_TRUE_X_IMP_X
     THEN IMP_RES_TAC NEW_STATE_PD_FROM_TI'_TO_T'SACK_1
     THEN RES_TAC
     THEN REWRITE_ASSUM_TAC ("New_State_Is_PD s' e' t'",[New_State_Is_PD])
     THEN IMP_RES_TAC I_last_ISO
     THEN ASM_REWRITE_TAC
            [WIRE;SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
  ]
 );;

let I_LAST_STABLE_HI_FROM_TI'_TO_T'RDY0 = TAC_PROOF
   (([],
   "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
      (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
      (tp' ti' t'rdy0 :timeC) .
    (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
     (VAL 1 (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0)) > 0) /\
     STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (ti'+1,t'rdy0)) ==>
        STABLE_HI (bsig I_last_O p') (ti'+1,t'rdy0)"),
   REWRITE_TAC [bsig;BSel;STABLE_HI]
   THEN REPEAT STRIP_TAC
   THEN IMP_RES_TAC (REWRITE_RULE [bsig;BSel] IB_READY_ASSUMPS)
   THENL [
      % Subgoal 1: "(ti' + 1) <= t'rdy0" %
      REWRITE_ASSUM_TAC
         ("STABLE_TRUE_THEN_FALSE(\t. SND(I_srdy_E(e' t)))(ti' + 1,t'rdy0)",
          [STABLE_TRUE_THEN_FALSE])
      THEN ASM_REWRITE_TAC[]
   ;
      % Subgoal 2:
        "(\t. SND(I_last_O(p' t)))t' = HI"
        [ "STABLE_TRUE_THEN_FALSE(\t. SND(I_srdy_E(e' t)))(ti' + 1,t'rdy0)" ]
        [ "(ti' + 1) <= t'" ]
        [ "t' <= t'rdy0" ] %
      BETA_TAC
      THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
      THEN IMP_RES_TAC (REWRITE_RULE [bsig;BSel] SACK_SIG_FALSE_DURING_DATA_0)
      THEN IMP_RES_TAC NTH_TRANS_CAUSAL
      THEN POP_ASSUM (\thm. ALL_TAC)
      THEN IMP_RES_TAC
              (SPECL ["tp':timeC";"ti':timeC";"1"] (RIMP LESS_EQ_MONO_ADD_EQ))
      THEN IMP_RES_TAC LESS_EQ_TRANS
      THEN IMP_RES_TAC
              (REWRITE_RULE [bsig;BSel] P_SIZE_STABLE_FROM_TP'_TO_T'RDY0)
      THEN IMP_RES_TAC
              (REWRITE_RULE
                 [bsig;BSel] P_DOWN_STABLE_FALSE_THEN_TRUE_FROM_TP'_TO_T'RDY0)
      THEN NRULE_ASSUM_TAC
              ("STABLE_FALSE_THEN_TRUE(\u'. P_downS(s' u'))(tp' + 1,t'rdy0 + 1)",
               (BETA_RULE o (REWRITE_RULE [STABLE_FALSE_THEN_TRUE])))
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN SPEC_ASSUM_TAC
              ("!t. (tp' + 1) <= t /\ t < (t'rdy0 + 1) ==> ~P_downS(s' t)",
               "t':timeC")
      THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LE_IMP_LT_SUC)
      THEN IMP_RES_TAC NTH_TIME_TRUE_X_IMP_X
```

```
        THEN IMP_RES_TAC NEW_STATE_PD_FROM_TI'_TO_T'SACK_1
        THEN RES_TAC
        THEN REWRITE_ASSUM_TAC ("New_State_Is_PD s' e' t'",[New_State_Is_PD])
        THEN IMP_RES_TAC I_last_ISO
        THEN ASM_REWRITE_TAC
               [WIRE;SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
        THEN IMP_RES_TAC GT_IMP_NOT_EQ
        THEN ASSUME_TAC
              (SPECL ["1";"SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0)"] MAXWORD)
        THEN RULE_ASSUM_TAC REDUCE_RULE
        THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LT_IMP_LE_PRE)
        THEN RULE_ASSUM_TAC REDUCE_RULE
        THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "0<=3"))
        THEN IMP_RES_TAC WORDN_1_NOT_EQUAL
        THEN ASSUME_TAC (SPEC "SND(L_ad_inE(e' (tp':timeC)))" SIZE_SUBARRAY_1)
        THEN IMP_RES_TAC
              (REDUCE_RULE
               (SPEC "SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0)"
                      WORDN_VAL_IDENT_1))
        THEN ASM_REWRITE_ASSUM_TAC
               ("~(WORDN 1(VAL 1(SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0))) =
                  WORDN 1 0)",[])
        THEN ASM_REWRITE_TAC[]
      ]
     );;

let I_SRDY_STABLE_TRUE_THEN_FALSE_FROM_T'RDY0_TO_T'RDY1 = TAC_PROOF
    (([],
     "! (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
        (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
        (tp' ti' t'rdy0 :timeC) .
      (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
       NTH_TIME_FALSE 0 (bsig I_srdy_E e') (ti'+1) t'rdy0 /\
       VAL 1 (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0)) > 0) ==>
          (?t'rdy1. STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e')(t'rdy0+1,t'rdy1))"),
     REPEAT STRIP_TAC
     THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
     THEN IMP_RES_TAC IB_READY_ASSUMPS
     THEN NRULE_ASSUM_TAC
            ("!u'. rdy_sig_ib e' p' u' ==>
                   (?v'. STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(u' + 1,v'))",
             (BETA_RULE o
              (SPEC "t'rdy0:timeC") o (REWRITE_RULE [rdy_sig_ib;BSel])))
     THEN IMP_RES_TAC NTH_TIME_FALSE_X_IMP_NOT_X
     THEN NRULE_ASSUM_TAC
            ("~bsig I_srdy_E e' t'rdy0",(BETA_RULE o (REWRITE_RULE [bsig;BSel])))
     THEN REWRITE_ASSUM_TAC
            ("NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)t'rdy0",[NTH_TIME_FALSE])
     THEN IMP_RES_TAC I_LAST_STABLE_HI_FROM_TI'_TO_T'RDY0
     THEN REWRITE_ASSUM_TAC
            ("STABLE_HI(bsig I_last_O p')(ti' + 1,t'rdy0)",[STABLE_HI;bsig;BSel])
     THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
     THEN NRULE_ASSUM_TAC
            ("!t. (ti' + 1) <= t /\ t <= t'rdy0 ==>
                  ((\t. SND(I_last_O(p' t)))t = HI)",
             (BETA_RULE o (SPEC "t'rdy0:timeC")))
     THEN ASSUME_TAC (SPEC "t'rdy0:timeC" LESS_EQ_REFL)
     THEN RES_TAC
     THEN RES_TAC
     THEN EXISTS_TAC "v':timeC"
     THEN ASM_REWRITE_TAC[]
    );;

let I_SRDY_TRUE_IMP_SACK_SIG_NOT_TRUE = TAC_PROOF
   (([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' t' u' :timeC) .
     (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
      STABLE_TRUE_THEN_FALSE (\t. SND(I_srdy_E(e' t)))(t'+1,u') /\
      STABLE_FALSE (Sack_Sig_Is_TRUE s' e') (t',t') /\
      1 <= u') ==>
```

```
        STABLE_FALSE (Sack_Sig_Is_TRUE s' e') (t',u'-1)"),
    REWRITE_TAC [STABLE_TRUE_THEN_FALSE;STABLE_FALSE;Sack_Sig_Is_TRUE]
    THEN BETA_TAC
    THEN REPEAT STRIP_TAC
    THENL [
       % Subgoal 1: "t' <= (u' - 1)" %
       IMP_RES_TAC (SPECL ["t'+1";"u':timeC";"1"] LESS_EQ_MONO_SUB)
       THEN RULE_ASSUM_TAC (REWRITE_RULE [SPECL ["t':timeC";"1"] ADD_SUB])
       THEN ASM_REWRITE_TAC[]
    ;
       % Subgoal 2: [ "~SND(I_srdy_E(e' t''))" ] %
       SPEC_ASSUM_TAC
         ("!t. (t' + 1) <= t /\ t < u' ==> SND(I_srdy_E(e' t))","t'':timeC")
       THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
       THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LE_PRE_IMP_LT)
       THEN ASM_CASES_TAC "(t'+1)<=t''"
       THENL [
          % Subgoal 2.1: [ "(t' + 1) <= t''" ] %
          RES_TAC
          THEN RES_TAC
       ;
          % Subgoal 2.2: [ "~(t' + 1) <= t''" ] %
          IMP_RES_TAC NOT_LESS_EQ_LESS
          THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LT_SUC_IMP_LE)
          THEN SUBGOAL_THEN
                 "t'' = (t':timeC)" (\thm. RULE_ASSUM_TAC (REWRITE_RULE [thm]))
          THENL [
            % Subgoal 2.2.1: (New subgoal) %
            IMP_RES_TAC LESS_EQUAL_ANTISYM
          ;
            % Subgoal 2.2.2: (Continue) %
            NRULE_ASSUM_TAC
                 ("!t. t' <= t /\ t <= t' ==>
               ~((P_sizeS(s' t) = (P_downS(s' t) => WORDN 1 1 | WORDN 1 0)) /\                ~SND(I_s-
rdy_E(e' t)) /\ New_State_Is_PD s' e' t)",
                  ((REWRITE_RULE [SPEC "t':timeC" LESS_EQ_REFL;
                                  DE_MORGAN_THM]) o (SPEC "t':timeC")))
            THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
            THEN RES_TAC
          ]
       ]
    ]
   );;

let I_LAST_HI_IMP_SACK_SIG_NOT_TRUE = TAC_PROOF
  (([],
   "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
      (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
      (tp' ti' t' :timeC) .
    (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
     (SND(I_last_O (p' t')) = HI)) ==>
        ~Sack_Sig_Is_TRUE s' e' t'"),
    REWRITE_TAC [Sack_Sig_Is_TRUE]
    THEN BETA_TAC
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
    THEN UNDISCH_TAC "SND(I_last_O(p' (t':timeC))) = HI"
    THEN IMP_RES_TAC I_last_ISO
    THEN ASM_REWRITE_TAC [WIRE]
    THEN COND_CASES_TAC
    THEN ASM_REWRITE_TAC [SYM_RULE (prove_constructors_distinct wire)]
   );;

let SACK_SIG_FALSE_DURING_DATA_0_1 = TAC_PROOF
    (([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (ti' t'sack t'rdy0 :timeC) .
     (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
      NTH_TIME_FALSE 0 (bsig I_srdy_E e') (ti'+1) t'rdy0 /\
      VAL 1 (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0)) > 0 /\
      STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (t'rdy0+1,t'rdy1)) ==>
```

```
          STABLE_FALSE (Sack_Sig_Is_TRUE s' e') (ti',t'rdy1-1)"),
     REWRITE_TAC [bsig;BSel]
     THEN REPEAT STRIP_TAC
     THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
     THEN IMP_RES_TAC
            (REWRITE_RULE [bsig;BSel]
                          I_SRDY_STABLE_TRUE_THEN_FALSE_FROM_T'RDY0_TO_T'RDY1)
     THEN REWRITE_ASSUM_TAC
            ("NTH_TIME_FALSE 0(\t. SND(I_srdy_E(e' t)))(ti' + 1)t'rdy0",
             [NTH_TIME_FALSE])
     THEN IMP_RES_TAC (REWRITE_RULE [bsig;BSel] SACK_SIG_FALSE_DURING_DATA_0)
     THEN IMP_RES_TAC
            (REWRITE_RULE [bsig;BSel] I_LAST_STABLE_HI_FROM_TI'_TO_T'RDY0)
     THEN SUBGOAL_THEN
            "STABLE_FALSE(Sack_Sig_Is_TRUE s' e')(t'rdy0,t'rdy0)" ASSUME_TAC
     THENL [
        % Subgoal 1: (New Subgoal) %
        REWRITE_TAC [STABLE_FALSE]
        THEN BETA_TAC
        THEN ASSUME_TAC (SPEC "t'rdy0:timeC" LESS_EQ_REFL)
        THEN ASM_REWRITE_TAC[]
        THEN REPEAT STRIP_TAC
        THEN SUBGOAL_THEN
               "t' = (t'rdy0:timeC)" (\thm. RULE_ASSUM_TAC (REWRITE_RULE [thm]))
        THENL [
           % Subgoal 1.1: (New Subgoal) %
           IMP_RES_TAC LESS_EQUAL_ANTISYM
        ;
           % Subgoal 1.2: (Continue) %
           NRULE_ASSUM_TAC
             ("STABLE_HI(\t:timeC. SND(I_last_O(p' t)))(ti' + 1,t'rdy0)",
              (BETA_RULE o (REWRITE_RULE [STABLE_HI])))
           THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
           THEN SPEC_ASSUM_TAC
                  ("!t. (ti'+1)<=t /\ t<=t'rdy0 ==> (SND(I_last_O(p' t))=HI)",
                   "t'rdy0:timeC")
           THEN RES_TAC
           THEN IMP_RES_TAC I_LAST_HI_IMP_SACK_SIG_NOT_TRUE
        ]
     ;
        % Subgoal 2: (Continue) %
        SUBGOAL_THEN "1 <= t'rdy1" ASSUME_TAC
        THENL [
           % Subgoal 2.1: (New Subgoal) %
           IMP_RES_TAC (RIMP ONE_LESS_EQ)
           THEN REWRITE_ASSUM_TAC
                 ("STABLE_TRUE_THEN_FALSE(\t. SND(I_srdy_E(e' t)))(ti'+1,t'rdy0)",
                  [STABLE_TRUE_THEN_FALSE])
           THEN REWRITE_ASSUM_TAC
                  ("STABLE_TRUE_THEN_FALSE(\t. SND(I_srdy_E(e' t)))
                    (t'rdy0+1,t'rdy1)",[STABLE_TRUE_THEN_FALSE])
           THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
           THEN ASSUME_TAC (SPECL ["ti':timeC";"1"] LESS_EQ_ADD)
           THEN ASSUME_TAC (SPECL ["t'rdy0:timeC";"1"] LESS_EQ_ADD)
           THEN IMP_RES_TAC LESS_EQ_TRANS
        ;
           % Subgoal 2.2: (Continue) %
           IMP_RES_TAC I_SRDY_TRUE_IMP_SACK_SIG_NOT_TRUE
           THEN SUBGOAL_THEN
                  "(ti' <= (t'rdy1-1)) /\
                   (t'rdy0 <= (t'rdy0-1)+1)"
                  STRIP_ASSUME_TAC
           THENL [
             % Subgoal 2.2.1: (New subgoal) %
             REWRITE_ASSUM_TAC
               ("STABLE_FALSE(Sack_Sig_Is_TRUE s' e')(ti',t'rdy0 - 1)",
                [STABLE_FALSE])
             THEN REWRITE_ASSUM_TAC
                    ("STABLE_TRUE_THEN_FALSE (\t. SND(I_srdy_E(e' t)))
                     (t'rdy0 + 1,t'rdy1)",[STABLE_TRUE_THEN_FALSE])
             THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
             THEN IMP_RES_TAC
```

```
                (SPECL ["t'rdy0+1";"t'rdy1:timeC";"1"] LESS_EQ_MONO_SUB)
         THEN REWRITE_ASSUM_TAC
                ("((t'rdy0 + 1) - 1) <= (t'rdy1 - 1)",[ADD_SUB])
         THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
         THEN ASSUME_TAC (SPECL ["t'rdy0:timeC";"1"] SUB_LESS_EQ)
         THEN IMP_RES_TAC LESS_EQ_TRANS
         THEN IMP_RES_TAC (SPECL ["t'rdy0:timeC";"1"] SUB_ADD)
         THEN IMP_RES_TAC
                (SPECL ["ti'+1";"t'rdy0:timeC";"1"] LESS_EQ_MONO_SUB)
         THEN REWRITE_ASSUM_TAC
                ("((ti' + 1) - 1) <= (t'rdy0 - 1)",[ADD_SUB])
         THEN IMP_RES_TAC LESS_EQ_TRANS
         THEN ASSUME_TAC (SPEC "t'rdy0:timeC" LESS_EQ_REFL)
         THEN ASM_REWRITE_TAC[]
        ;
         % Subgoal 2.2.2: (Continue) %
         IMP_RES_TAC SUP_INTERVAL_STABLE_FALSE
         THEN ASM_REWRITE_TAC[]
        ]
      ]
   ]
  );;

let P_DOWN_TRUE_THEN_STABLE_FALSE_FROM_T'RDY0_TO_T'RDY1 = TAC_PROOF
   (([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' t'rdy0 t'rdy1 :timeC) .
    (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
     NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)t'rdy0 /\
     STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(t'rdy0 + 1,t'rdy1) /\
     (VAL 1(SUBARRAY(SND(L_ad_inE(e' tp')))(1,0))) > 0) ==>
        TRUE_THEN_STABLE_FALSE(\u'. P_downS(s' u'))(t'rdy0 + 1,t'rdy1)"),
    REWRITE_TAC [TRUE_THEN_STABLE_FALSE]
    THEN BETA_TAC
    THEN REPEAT STRIP_TAC
    THENL [
       % Subgoal 1: "(t'rdy0 + 1) <= t'rdy1" %
       REWRITE_ASSUM_TAC
         ("STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(t'rdy0 + 1,t'rdy1)",
          [STABLE_TRUE_THEN_FALSE])
       THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
       THEN ASM_REWRITE_TAC[]
    ;
       % Subgoal 2: "P_downS(s'(t'rdy0 + 1))" %
       REWRITE_ASSUM_TAC
         ("NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)t'rdy0",[NTH_TIME_FALSE])
       THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
       THEN IMP_RES_TAC P_DOWN_STABLE_FALSE_THEN_TRUE_FROM_TP'_TO_T'RDY0
       THEN NRULE_ASSUM_TAC
              ("STABLE_FALSE_THEN_TRUE(\u'. P_downS(s' u'))(tp' + 1,t'rdy0 + 1)",
               (BETA_RULE o (REWRITE_RULE [STABLE_FALSE_THEN_TRUE])))
       THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
       THEN ASM_REWRITE_TAC[]
    ;
       % Subgoal 3:  [ "(t'rdy0 + 1) < t'" ]
                     [ "t' <= t'rdy1" ]
                     [ "P_downS(s' t')" ] %
       UNDISCH_TAC "P_downS(s' (t':timeC))"
       THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
       THEN SUBGOAL_THEN "t' = (t'-1)+1" (\thm. PURE_ONCE_REWRITE_TAC [thm])
       THENL [
          % Subgoal 3.1: (New Subgoal) %
          IMP_RES_TAC (RIMP ONE_LESS_EQ)
          THEN REWRITE_ASSUM_TAC
                 ("NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)t'rdy0",
                  [NTH_TIME_FALSE])
          THEN REWRITE_ASSUM_TAC
                 ("STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(ti' + 1,t'rdy0)",
                  [STABLE_TRUE_THEN_FALSE])
          THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
          THEN ASSUME_TAC (SPECL ["ti':timeC";"1"] LESS_EQ_ADD)
          THEN ASSUME_TAC (SPECL ["t'rdy0:timeC";"1"] LESS_EQ_ADD)
          THEN IMP_RES_TAC LT_IMP_LE
          THEN IMP_RES_TAC LESS_EQ_TRANS
          THEN IMP_RES_TAC LESS_EQ_TRANS
          THEN IMP_RES_TAC (SPECL ["t':timeC";"1"] (SYM_RULE SUB_ADD))
       ;
          % Subgoal 3.2: (Continue) %
          IMP_RES_TAC P_down_ISO
          THEN ASM_REWRITE_TAC[]
          THEN NRULE_ASSUM_TAC
                ("STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(t'rdy0 + 1,t'rdy1)",
                 (BETA_RULE o (REWRITE_RULE [STABLE_TRUE_THEN_FALSE;bsig;BSel])))
          THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
          THEN SPEC_ASSUM_TAC
                 ("!t. (t'rdy0 + 1) <= t /\ t < t'rdy1 ==> SND(I_srdy_E(e' t))",
                  "t'-1")
          THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LT_IMP_LE_PRE)
          THEN SUBGOAL_THEN "1 <= t'" ASSUME_TAC
          THENL [
            % Subgoal 3.2.1: "1 <= t'" %
            IMP_RES_TAC (RIMP ONE_LESS_EQ)
            THEN ASSUME_TAC (SPECL ["ti':timeC";"1"] LESS_EQ_ADD)
            THEN REWRITE_ASSUM_TAC
                   ("NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)t'rdy0",
                    [NTH_TIME_FALSE])
            THEN REWRITE_ASSUM_TAC
                   ("STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(ti' + 1,t'rdy0)",
                    [STABLE_TRUE_THEN_FALSE])
            THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
            THEN ASSUME_TAC (SPECL ["t'rdy0:timeC";"1"] LESS_EQ_ADD)
            THEN ASSUME_TAC (SPECL ["t':timeC";"1"] SUB_LESS_EQ)
            THEN IMP_RES_TAC LESS_EQ_TRANS
            THEN IMP_RES_TAC LESS_EQ_TRANS
          ;
            % Subgoal 3.2.2: (Continue) %
            IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LE_IMP_PRE_LT)
            THEN RES_TAC
            THEN ASM_REWRITE_TAC[]
          ]
       ]
    ]
   );;

let OFFSET_P_SIZE_STABLE_FROM_T'RDY0_TO_T'RDY1 = TAC_PROOF
   (([],
    "! (u' :timeC)
       (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' t'rdy0 t'rdy1 :timeC) .
     (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
      NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)t'rdy0 /\
      STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (t'rdy0+1,t'rdy1) /\
      VAL 1 (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0)) > 0 /\
      ((t'rdy0+u'+2) <= t'rdy1+1))
      ==>
     (P_sizeS (s' (t'rdy0+u'+2)) =
        DECN 1 (SUBARRAY (SND(L_ad_inE(e' tp'))) (1,0)))"),
    INDUCT_TAC
    THEN REWRITE_TAC [ADD1;ADD_CLAUSES;ADD_ASSOC;SYM_RULE (REDUCE_CONV "1+1")]
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
    THENL [
       % Subgoal 1: (Base Case) %
       IMP_RES_TAC P_size_ISO
       THEN ASM_REWRITE_TAC[]
       THEN IMP_RES_TAC SACK_SIG_FALSE_DURING_DATA_0_1
       THEN IMP_RES_TAC P_LOAD_TRUE_THEN_STABLE_FALSE_FROM_TP'_TO_T'SACK
       THEN SUBGOAL_THEN
              "tp' < t'rdy0 /\ (t'rdy0+1) <= t'rdy1" STRIP_ASSUME_TAC
       THENL [
          % Subgoal 1.1: (New Subgoal) %
          IMP_RES_TAC NTH_TRANS_CAUSAL
      THEN POP_ASSUM (\thm. ALL_TAC)
      THEN ASSUME_TAC (SPEC "ti':timeC" (REWRITE_RULE [ADD1] LESS_SUC_REFL))
      THEN REWRITE_ASSUM_TAC
             ("NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)t'rdy0",
              [NTH_TIME_FALSE;STABLE_TRUE_THEN_FALSE])
      THEN REWRITE_ASSUM_TAC
             ("STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(t'rdy0 + 1,t'rdy1)",
              [STABLE_TRUE_THEN_FALSE])
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN IMP_RES_TAC LESS_EQ_TRANS
      THEN IMP_RES_TAC LESS_EQ_LESS_TRANS
      THEN IMP_RES_TAC LESS_LESS_EQ_TRANS
      THEN IMP_RES_TAC LT_IMP_LE
      THEN ASM_REWRITE_TAC[]
   ;
      % Subgoal 1.2: (Continue) %
      ASSUME_TAC (SPECL ["t'rdy0:timeC";"1"] LESS_EQ_ADD)
      THEN IMP_RES_TAC LESS_LESS_EQ_TRANS
      THEN IMP_RES_TAC LESS_EQ_TRANS
      THEN NRULE_ASSUM_TAC
             ("TRUE_THEN_STABLE_FALSE(\u'. P_loadS(s' u'))(tp',t'rdy1)",
              (BETA_RULE o (REWRITE_RULE [TRUE_THEN_STABLE_FALSE])))
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN SPEC_ASSUM_TAC
             ("!t. tp' < t /\ t <= t'rdy1 ==> ~P_loadS(s' t)","t'rdy0:timeC")
      THEN IMP_RES_TAC P_LOAD_TRUE_THEN_STABLE_FALSE_FROM_TP'_TO_T'SACK
      THEN NRULE_ASSUM_TAC
             ("TRUE_THEN_STABLE_FALSE(\u'. P_loadS(s' u'))(tp',t'rdy1)",
              (BETA_RULE o (REWRITE_RULE [TRUE_THEN_STABLE_FALSE])))
      THEN SPEC_ASSUM_TAC
             ("!t. tp' < t /\ t <= t'rdy1 ==> ~P_loadS(s' t)","t'rdy0+1")
      THEN RES_TAC
      THEN ASM_REWRITE_TAC[]
      THEN REWRITE_ASSUM_TAC
             ("NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)t'rdy0",
              [NTH_TIME_FALSE])
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN IMP_RES_TAC P_DOWN_STABLE_FALSE_THEN_TRUE_FROM_TP'_TO_T'RDY0
      THEN NRULE_ASSUM_TAC
             ("STABLE_FALSE_THEN_TRUE(\u'. P_downS(s' u'))(tp'+1,t'rdy0+1)",
              (BETA_RULE o (REWRITE_RULE [STABLE_FALSE_THEN_TRUE])))
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN SPEC_ASSUM_TAC
             ("!t. (tp' + 1) <= t /\ t < (t'rdy0 + 1) ==> ~P_downS(s' t)",
              "t'rdy0:timeC")
      THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LT_IMP_SUC_LE)
      THEN ASSUME_TAC
             (SPEC "t'rdy0:timeC" (REWRITE_RULE [ADD1] LESS_SUC_REFL))
      THEN RES_TAC
      THEN ASSUME_TAC (SPEC "t'rdy0:timeC" LESS_EQ_REFL)
      THEN IMP_RES_TAC P_SIZE_STABLE_FROM_TP'_TO_T'RDY0
      THEN ASM_REWRITE_TAC[]
   ]
;
   % Subgoal 2: (Induction Step) %
   IMP_RES_TAC P_size_ISO
   THEN ONCE_ASM_REWRITE_TAC[]
   THEN POP_ASSUM (\thm. ALL_TAC) %KEEP%
   THEN RULE_ASSUM_TAC
          (\thm. REWRITE_RULE [SYM_RULE (REDUCE_CONV "1+1");ADD_ASSOC] thm)
   THEN ASSUME_TAC (SPECL ["(((t'rdy0 + u') + 1) + 1)";"1"] LESS_EQ_ADD)
   THEN IMP_RES_TAC LESS_EQ_TRANS
   THEN RES_TAC
   THEN IMP_RES_TAC SACK_SIG_FALSE_DURING_DATA_0_1
   THEN IMP_RES_TAC P_LOAD_TRUE_THEN_STABLE_FALSE_FROM_TP'_TO_T'SACK
   THEN NRULE_ASSUM_TAC
          ("TRUE_THEN_STABLE_FALSE(\u'. P_loadS(s' u'))(tp',t'rdy1)",
           (BETA_RULE o (REWRITE_RULE [TRUE_THEN_STABLE_FALSE])))
   THEN IMP_RES_TAC P_DOWN_TRUE_THEN_STABLE_FALSE_FROM_T'RDY0_TO_T'RDY1
   THEN NRULE_ASSUM_TAC
          ("TRUE_THEN_STABLE_FALSE(\u'. P_downS(s' u'))(t'rdy0 + 1,t'rdy1)",
           (BETA_RULE o (REWRITE_RULE [TRUE_THEN_STABLE_FALSE])))
```

```
        THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
        THEN SPEC_ASSUM_TAC
               ("!t. tp' < t /\ t <= t'rdy1 ==> ~P_loadS(s' t)",
                "((t'rdy0+u')+1)+1")
        THEN SPEC_ASSUM_TAC
               ("!t. (t'rdy0 + 1) < t /\ t <= t'rdy1 ==> ~P_downS(s' t)",
                "((t'rdy0+u')+1)+1")
        THEN ASSUME_TAC (SPECL ["t'rdy0:timeC";"u':timeC"] LESS_EQ_ADD)
        THEN IMP_RES_TAC
               (RIMP (SPECL ["t'rdy0:timeC";"t'rdy0+u'";"1"] LESS_EQ_MONO_ADD_EQ))
        THEN ASSUME_TAC (SPEC "(t'rdy0+u')+1" (REWRITE_RULE [ADD1] LESS_SUC_REFL))
        THEN IMP_RES_TAC LESS_EQ_LESS_TRANS
        THEN IMP_RES_TAC
               (SPECL ["(((t'rdy0+u')+1)+1)+1";"t'rdy1+1";"1"] LESS_EQ_MONO_SUB)
        THEN REWRITE_ASSUM_TAC
               ("((((t'rdy0 + u') + 1) + 1) + 1) - 1) <= ((t'rdy1 + 1) - 1)",
                [ADD_SUB])
        THEN IMP_RES_TAC NTH_TRANS_CAUSAL
        THEN ASSUME_TAC (SPECL ["ti':timeC";"1"] LESS_EQ_ADD)
        THEN NRULE_ASSUM_TAC
               ("NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)t'rdy0",
                (BETA_RULE o
                 (REWRITE_RULE [NTH_TIME_FALSE;STABLE_TRUE_THEN_FALSE])))
        THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
        THEN ASSUME_TAC (SPECL ["t'rdy0+u'";"1"] LESS_EQ_ADD)
        THEN IMP_RES_TAC LESS_EQ_TRANS
        THEN IMP_RES_TAC LESS_EQ_LESS_TRANS
        THEN RES_TAC
        THEN ASM_REWRITE_TAC[]
     ]
    );;

let P_SIZE_STABLE_FROM_T'RDY0_TO_T'RDY1 = TAC_PROOF
   (([],
    "! (t' :timeC)
       (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' t'rdy0 t'rdy1 :timeC) .
     (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
      NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)t'rdy0 /\
      STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (t'rdy0+1,t'rdy1) /\
      VAL 1 (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0)) > 0 /\
      ((t'rdy0+2) <= t') /\
      (t' <= t'rdy1+1))
      ==>
     (P_sizeS (s' t') = DECN 1 (SUBARRAY (SND(L_ad_inE (e' tp'))) (1,0)))"),
    REWRITE_TAC [SYM_RULE (REDUCE_CONV "1+1");ADD_ASSOC]
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC (SPEC "t'-tp'+2" OFFSET_P_SIZE_STABLE_FROM_T'RDY0_TO_T'RDY1)
    THEN SPECL_ASSUM_TAC
           ("!(t' tp''':timeC). (t'rdy0+((t'-(tp'''+2))+2)) <= (t'rdy1 + 1) ==>
             (P_sizeS(s'(t'rdy0 + ((t' - (tp''' + 2)) + 2))) =
             DECN 1(SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0)))",
            ["t':timeC";"t'rdy0:timeC"])
    THEN SUBGOAL_THEN
            "(t'rdy0 + ((t' - (t'rdy0 + 2)) + 2)) = t'"
            (\thm. RULE_ASSUM_TAC (REWRITE_RULE [thm]))
      THENL [
         % Subgoal 1: -New subgoal- "t'rdy0 + ((t' - (t'rdy0 + 2)) + 2) = t'" %
         REWRITE_TAC [SYM_RULE (ASSOC_SUB_SUB1)]
         THEN SUBGOAL_THEN "2 <= (t' - t'rdy0)" ASSUME_TAC
         THENL [
            % Subgoal 1.1: "2 <= (t' - t'rdy0)"
                             [ "((t'rdy0 + 1) + 1) <= t'" ] %
            REWRITE_TAC
              [SYM_RULE (SPECL ["2";"t'-t'rdy0";"t'rdy0:timeC"]
                                 LESS_EQ_MONO_ADD_EQ)]
            THEN ASSUME_TAC (SPECL ["t'rdy0:timeC";"1"] LESS_EQ_ADD)
            THEN ASSUME_TAC (SPECL ["t'rdy0+1";"1"] LESS_EQ_ADD)
            THEN IMP_RES_TAC LESS_EQ_TRANS
            THEN IMP_RES_TAC (SPECL ["t':timeC";"t'rdy0:timeC"] SUB_ADD)
            THEN ASM_REWRITE_TAC[]
```

```
          THEN PURE_ONCE_REWRITE_TAC [ADD_SYM]
          THEN ASM_REWRITE_TAC[SYM_RULE (REDUCE_CONV "1+1");ADD_ASSOC]
        ;
          % Subgoal 1.2: [ "2 <= (t' - t'rdy0)" ] %
          ASSUME_TAC (SPEC "2" LESS_EQ_REFL)
          THEN IMP_RES_TAC (SPECL ["t'-t'rdy0";"2";"2"] ASSOC_SUB_ADD1)
          THEN ASM_REWRITE_TAC[SUB_EQUAL_0;ADD_CLAUSES]
          THEN PURE_ONCE_REWRITE_TAC [ADD_SYM]
          THEN ASSUME_TAC (SPECL ["t'rdy0:timeC";"1"] LESS_EQ_ADD)
          THEN ASSUME_TAC (SPECL ["t'rdy0+1";"1"] LESS_EQ_ADD)
          THEN IMP_RES_TAC LESS_EQ_TRANS
          THEN IMP_RES_TAC (SPECL ["t':timeC";"t'rdy0:timeC"] SUB_ADD)
        ]
     ;
       % Subgoal 2 %
       RES_TAC
     ]
    );;

let DECN_WORDN_1_NOT_EQ = mk_thm
   ([],
    "! (x :wordn) (m n :num) .
     (VAL 1 x > n) ==>
       (n > m) ==>
         ~(DECN 1 x = WORDN 1 m)"
   );;

let I_LAST_STABLE_HI_FROM_T'RDY0_TO_T'RDY1 = TAC_PROOF
   (([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' t'rdy0 :timeC) .
     (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
      NTH_TIME_FALSE 0 (bsig I_srdy_E e') (ti'+1) t'rdy0 /\
      STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (t'rdy0+1,t'rdy1) /\
      (VAL 1 (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0)) > 1)) ==>
         STABLE_HI (bsig I_last_O p') (t'rdy0+1,t'rdy1)"),
    REWRITE_TAC [bsig;BSel;STABLE_HI]
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC (REWRITE_RULE [bsig;BSel] IB_READY_ASSUMPS)
    THENL [
       % Subgoal 1: "(t'rdy0 + 1) <= t'rdy1" %
       REWRITE_ASSUM_TAC
          ("STABLE_TRUE_THEN_FALSE(\t. SND(I_srdy_E(e' t)))(t'rdy0 + 1,t'rdy1)",
           [STABLE_TRUE_THEN_FALSE])
       THEN ASM_REWRITE_TAC[]
     ;
       % Subgoal 2:
         "(\t. SND(I_last_O(p' t)))t' = HI"
         [ "STABLE_TRUE_THEN_FALSE(\t. SND(I_srdy_E(e' t)))(t'rdy0 + 1,t'rdy1)" ]
         [ "(t'rdy0 + 1) <= t'" ]
         [ "t' <= t'rdy1" ] %
       BETA_TAC
       THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
       THEN SUBGOAL_THEN
              "VAL 1 (SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0)) > 0"
              ASSUME_TAC
       THENL [
          % Subgoal 2.1: (New subgoal) %
          REWRITE_TAC [GREATER]
          THEN RULE_ASSUM_TAC (\thm. REWRITE_RULE [GREATER] thm)
          THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "0<1"))
          THEN IMP_RES_TAC LESS_TRANS
        ;
          % Subgoal 2.2: (Continue) %
          SUBGOAL_THEN "(tp'<=ti') /\ ((ti'+1) <= t'rdy0)" STRIP_ASSUME_TAC
          THENL [
            % Subgoal 2.2.1: (New subgoal) %
            IMP_RES_TAC NTH_TRANS_CAUSAL
            THEN REWRITE_ASSUM_TAC
                   ("NTH_TIME_FALSE 0(\t. SND(I_srdy_E(e' t)))(ti' + 1)t'rdy0",
                    [NTH_TIME_FALSE;STABLE_TRUE_THEN_FALSE])
     THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
     THEN ASM_REWRITE_TAC[]
    ;
     % Subgoal 2.2.2: (Continue) %
     ASSUME_TAC (SPECL ["t'rdy0:timeC";"1"] LESS_EQ_ADD)
     THEN ASSUME_TAC (SPECL ["t'rdy1:timeC";"1"] LESS_EQ_ADD)
     THEN IMP_RES_TAC LESS_EQ_TRANS
     THEN IMP_RES_TAC
            (REWRITE_RULE [bsig;BSel] SACK_SIG_FALSE_DURING_DATA_0_1)
     THEN IMP_RES_TAC NTH_TIME_TRUE_X_IMP_X
     THEN IMP_RES_TAC NEW_STATE_PD_FROM_TI'_TO_T'SACK_1
     THEN IMP_RES_TAC
            (REWRITE_RULE
               [bsig;BSel]
               P_DOWN_TRUE_THEN_STABLE_FALSE_FROM_T'RDY0_TO_T'RDY1)
     THEN NRULE_ASSUM_TAC
           ("TRUE_THEN_STABLE_FALSE(\u'.P_downS(s' u'))(t'rdy0+1,t'rdy1)",
            (BETA_RULE o (REWRITE_RULE [TRUE_THEN_STABLE_FALSE])))
     THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
     THEN SPEC_ASSUM_TAC
            ("!t. (t'rdy0 + 1) < t /\ t <= t'rdy1 ==> ~P_downS(s' t)",
             "t':timeC")
     THEN REWRITE_ASSUM_TAC ("New_State_Is_PD s' e' t'",[New_State_Is_PD])
     THEN IMP_RES_TAC I_last_ISO
     THEN ASM_REWRITE_TAC
            [WIRE;SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
     THEN POP_ASSUM (\thm. ALL_TAC)
     THEN ASM_CASES_TAC "(t'rdy0+1) < t'"
     THENL [
       % Subgoal 2.2.2.1: [ "(t'rdy0 + 1) < t'" ] %
       IMP_RES_TAC (REWRITE_RULE [ADD1] LT_IMP_SUC_LE)
       THEN NRULE_ASSUM_TAC
              ("((t'rdy0 + 1) + 1) <= t'",
               (REDUCE_RULE o (REWRITE_RULE [SYM_RULE ADD_ASSOC])))
       THEN IMP_RES_TAC
            (REWRITE_RULE [bsig;BSel] P_SIZE_STABLE_FROM_T'RDY0_TO_T'RDY1)
       THEN RES_TAC
       THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "1>0"))
       THEN IMP_RES_TAC
              (SPECL ["SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0)";
                      "0";"1"] DECN_WORDN_1_NOT_EQ)
       THEN ASM_REWRITE_TAC[]
      ;
       % Subgoal 2.2.2.2: [ "~(t'rdy0 + 1) < t'" ] %
       SUBGOAL_THEN "t' = t'rdy0+1" (\thm. REWRITE_TAC [thm])
       THENL [
         % Subgoal 2.2.2.2.1: (New Subgoal) %
         IMP_RES_TAC NOT_LESS
         THEN IMP_RES_TAC LESS_EQUAL_ANTISYM
        ;
         % Subgoal 2.2.2.2.2: (Continue) %
         REWRITE_ASSUM_TAC
           ("NTH_TIME_FALSE 0(\t. SND(I_srdy_E(e' t)))(ti' + 1)t'rdy0",
            [NTH_TIME_FALSE])
         THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
         THEN IMP_RES_TAC
                (REWRITE_RULE [bsig;BSel] P_SIZE_STABLE_AT_T'RDY0_PLUS_1)
         THEN IMP_RES_TAC GT_IMP_NOT_EQ
         THEN ASSUME_TAC
              (REDUCE_RULE
               (SPECL ["1";"SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0)"]
               MAXWORD))
         THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LT_IMP_LE_PRE)
         THEN NRULE_ASSUM_TAC
                ("(VAL 1(SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0)))
                   <= (4 - 1)",REDUCE_RULE)
         THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "1<=3"))
         THEN IMP_RES_TAC WORDN_1_NOT_EQUAL
         THEN ASSUME_TAC
                (SPEC "SND(L_ad_inE(e' (tp':timeC)))" SIZE_SUBARRAY_1)
         THEN IMP_RES_TAC WORDN_VAL_IDENT_1
         THEN ASM_REWRITE_ASSUM_TAC
```

```
                    ("~(WORDN 1(VAL 1(SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))
                       (1,0))) = WORDN 1 1)",[])
                 THEN ASM_REWRITE_TAC[]
               ]
             ]
           ]
        ]
     ]
    );;

let I_LAST_FOR_BLOCK_SIZE_1 = TAC_PROOF
   (([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' t'rdy0 :timeC) .
     (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
      (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0) = WORDN 1 1) /\
      NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)t'rdy0 /\
      STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (t'rdy0+1,t'rdy1)) ==>
         (STABLE_HI (bsig I_last_O p') (ti'+1,t'rdy0) /\
          STABLE_LO (bsig I_last_O p') (t'rdy0+1,t'rdy1))"),
    REPEAT STRIP_TAC
    THENL [
       % Subgoal 1: "STABLE_HI(bsig I_last_O p')(ti' + 1,t'rdy0)" %
       SUBGOAL_THEN
         "VAL 1(SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0)) > 0" ASSUME_TAC
       THENL [
          % Subgoal 1.1: (New Subgoal) %
          ASM_REWRITE_TAC[]
          THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "1 <= 3"))
          THEN IMP_RES_TAC VAL_WORDN_IDENT_1
          THEN ASM_REWRITE_TAC[]
          THEN REDUCE_TAC
       ;
          % Subgoal 1.2: (Continue) %
          REWRITE_ASSUM_TAC
            ("NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)t'rdy0",
             [NTH_TIME_FALSE])
          THEN IMP_RES_TAC I_LAST_STABLE_HI_FROM_TI'_TO_T'RDY0
       ]
    ;
       % Subgoal 2: "STABLE_LO(bsig I_last_O p')(t'rdy0 + 1,t'rdy1)" %
       SUBGOAL_THEN
         "VAL 1(SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0)) > 0" ASSUME_TAC
       THENL [
          % Subgoal 2.1: (New Subgoal) %
          ASM_REWRITE_TAC[]
          THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "1 <= 3"))
          THEN IMP_RES_TAC VAL_WORDN_IDENT_1
          THEN ASM_REWRITE_TAC[]
          THEN REDUCE_TAC
       ;
          % Subgoal 2.2: (Continue) %
          REWRITE_TAC [STABLE_LO;bsig;BSel]
          THEN BETA_TAC
          THEN REPEAT STRIP_TAC
          THENL [
            % Subgoal 2.2.1: "(t'rdy0 + 1) <= t'rdy1" %
            REWRITE_ASSUM_TAC
              ("STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(t'rdy0 + 1,t'rdy1)",
               [STABLE_TRUE_THEN_FALSE])
            THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
            THEN ASM_REWRITE_TAC[]
          ;
            % Subgoal 2.2.2: "SND(I_last_O(p' t')) = LO" %
            IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
            THEN IMP_RES_TAC I_last_ISO
            THEN ASM_REWRITE_TAC[]
            THEN POP_ASSUM (\thm. ALL_TAC)
            THEN IMP_RES_TAC P_SIZE_STABLE_FROM_T'RDY0_TO_T'RDY1
            THEN IMP_RES_TAC SACK_SIG_FALSE_DURING_DATA_0_1
            THEN IMP_RES_TAC NTH_TIME_TRUE_X_IMP_X
            THEN IMP_RES_TAC NEW_STATE_PD_FROM_TI'_TO_T'SACK_1
            THEN POP_ASSUM (\thm. ALL_TAC)
            THEN SPEC_ASSUM_TAC
                   ("!t'. (ti' + 1) <= t' ==> t' <= t'rdy1 ==>
                          New_State_Is_PD s' e' t'","t':timeC")
            THEN SUBGOAL_THEN "(ti'+1)<=t'" ASSUME_TAC
            THENL [
              % Subgoal 2.2.2.1: (New Subgoal) %
              REWRITE_ASSUM_TAC
                ("NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)t'rdy0",
                 [NTH_TIME_FALSE;STABLE_TRUE_THEN_FALSE])
              THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
              THEN ASSUME_TAC (SPECL ["t'rdy0:timeC";"1"] LESS_EQ_ADD)
              THEN IMP_RES_TAC LESS_EQ_TRANS
            ;
              % Subgoal 2.2.2.2: (Continue) %
              RES_TAC
              THEN REWRITE_ASSUM_TAC
                     ("New_State_Is_PD s' e' t'",[New_State_Is_PD])
              THEN ASM_REWRITE_TAC
                     [SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
              THEN IMP_RES_TAC
                     P_DOWN_TRUE_THEN_STABLE_FALSE_FROM_T'RDY0_TO_T'RDY1
              THEN NRULE_ASSUM_TAC
                   ("TRUE_THEN_STABLE_FALSE(\u'.P_downS(s' u'))(t'rdy0+1,t'rdy1)",
                     (BETA_RULE o (REWRITE_RULE [TRUE_THEN_STABLE_FALSE])))
              THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
              THEN SPEC_ASSUM_TAC
                     ("!t. (t'rdy0 + 1) < t /\ t <= t'rdy1 ==> ~P_downS(s' t)",
                      "t':timeC")
              THEN ASM_CASES_TAC "(t'rdy0 + 1) < t'"
              THENL [
                % Subgoal 2.2.2.2.1: [ "(t'rdy0 + 1) < t'" ] %
                IMP_RES_TAC (REWRITE_RULE [ADD1] LT_IMP_SUC_LE)
                THEN REWRITE_ASSUM_TAC
                       ("((t'rdy0+1)+1)<=t'",[ASSOC_ADD_ADD1;REDUCE_CONV "1+1"])
                THEN ASSUME_TAC (SPECL ["t'rdy1:timeC";"1"] LESS_EQ_ADD)
                THEN IMP_RES_TAC LESS_EQ_TRANS
                THEN RES_TAC
                THEN ASM_REWRITE_TAC[]
                THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "1<=3"))
                THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "0<=3"))
                THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "1=0+1"))
                THEN IMP_RES_TAC (SPECL ["1";"0"] DECN_WORDN_1)
                THEN DELETE_ASSUM_TAC "1 = 0 + 1"
                THEN ASM_REWRITE_TAC [WIRE]
              ;
                % Subgoal 2.2.2.2.2: [ "~(t'rdy0 + 1) < t'" ] %
                IMP_RES_TAC NOT_LESS
                THEN SUBGOAL_THEN "t' = t'rdy0+1" (\thm. REWRITE_TAC [thm])
                THENL [
                  % Subgoal 2.2.2.2.2.1: (New Subgoal) %
                  IMP_RES_TAC LESS_EQUAL_ANTISYM
                ;
                  % Subgoal 2.2.2.2.2.2: (Continue) %
                  REWRITE_ASSUM_TAC
                    ("NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)t'rdy0",
                     [NTH_TIME_FALSE])
                  THEN IMP_RES_TAC P_SIZE_STABLE_AT_T'RDY0_PLUS_1
                  THEN ASM_REWRITE_TAC [WIRE]
                ]
              ]
            ]
          ]
        ]
      ]
    );;

let I_SRDY_STABLE_TRUE_THEN_FALSE_FROM_T'RDY1_TO_T'RDY2 = TAC_PROOF
  (([],
   "! (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
      (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
      (tp' ti' t'rdyl :timeC) .
     (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
      NTH_TIME_FALSE 1 (bsig I_srdy_E e') (ti'+1) t'rdy1 /\
      VAL 1 (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0)) > 1) ==>
         (?t'rdy2. STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e')(t'rdy1+1,t'rdy2))"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
    THEN IMP_RES_TAC IB_READY_ASSUMPS
    THEN NRULE_ASSUM_TAC
           ("!u'. rdy_sig_ib e' p' u' ==>
                  (?v'. STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(u'-+ 1,v'))",
            (BETA_RULE o
             (SPEC "t'rdy1:timeC") o (REWRITE_RULE [rdy_sig_ib;BSel])))
    THEN IMP_RES_TAC NTH_TIME_FALSE_X_IMP_NOT_X
    THEN NRULE_ASSUM_TAC
           ("~bsig I_srdy_E e' t'rdy1",(BETA_RULE o (REWRITE_RULE [bsig;BSel])))
    THEN SUBGOAL_THEN
           "NTH_TIME_FALSE 0 (bsig I_srdy_E e') (ti'+1) u' /\
            STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (u'+1,t'rdy1)"
            STRIP_ASSUME_TAC
    THENL [
      % Subgoal 1: (New Subgoal) %
      REWRITE_ASSUM_TAC
        ("NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti' + 1)t'rdy1",
         [num_CONV "1";NTH_TIME_FALSE])
      THEN CHOOSE_ASSUM_TAC
           "?t. STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(ti'+(SUC 0),t)/\
                STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(t+(SUC 0),t'rdy1)"
      THEN POP_ASSUM_LIST
             (MAP_EVERY
              (\thm. STRIP_ASSUME_TAC
                      (REWRITE_RULE [ADD1;ADD_CLAUSES] thm)))
      THEN SUBGOAL_THEN "u' = (t':timeC)" (\thm. REWRITE_TAC [thm])
      THENL [
        % Subgoal 1.1: (New subgoal) %
        IMP_RES_TAC STABLE_TRUE_THEN_FALSE_UNIQUE
      ;
        % Subgoal 1.2: (Continue) %
        ASM_REWRITE_TAC [NTH_TIME_FALSE]
      ]
    ;
      % Subgoal 2: (Continue) %
      IMP_RES_TAC I_LAST_STABLE_HI_FROM_T'RDY0_TO_T'RDY1
      THEN NRULE_ASSUM_TAC
             ("STABLE_HI(bsig I_last_O p')(u' + 1,t'rdy1)",
              (BETA_RULE o (REWRITE_RULE [STABLE_HI])))
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN NRULE_ASSUM_TAC
             ("!t. (u' + 1) <= t /\ t <= t'rdy1 ==> (bsig I_last_O p' t = HI)",
              (BETA_RULE o (SPEC "t'rdy1:timeC") o (REWRITE_RULE [bsig;BSel])))
      THEN ASSUME_TAC (SPEC "t'rdy1:timeC" LESS_EQ_REFL)
      THEN RES_TAC
      THEN RES_TAC
      THEN EXISTS_TAC "v':timeC"
      THEN ASM_REWRITE_TAC[]
    ]
   );;

let SACK_SIG_FALSE_DURING_DATA_1_2 = TAC_PROOF
   (([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (ti' t'rdy1 t'rdy2 :timeC) .
     (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
      NTH_TIME_FALSE 1 (bsig I_srdy_E e') (ti'+1) t'rdy1 /\
      VAL 1 (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0)) > 1 /\
      STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (t'rdy1+1,t'rdy2)) ==>
         STABLE_FALSE (Sack_Sig_Is_TRUE s' e') (ti',t'rdy2-1)"),
    REWRITE_TAC [bsig;BSel]
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
    THEN SUBGOAL_THEN
```

```
        "VAL 1 (SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0)) > 0"
        ASSUME_TAC
THENL [
   % Subgoal 1: (New subgoal) %
   REWRITE_TAC [GREATER]
   THEN RULE_ASSUM_TAC (\thm. REWRITE_RULE [GREATER] thm)
   THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "0<1"))
   THEN IMP_RES_TAC LESS_TRANS
;
   % Subgoal 2: (Continue) %
   ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "0<1"))
   THEN IMP_RES_TAC PRIOR_FALSE_EVENTS_EXIST
   THEN SUBGOAL_THEN
          "STABLE_TRUE_THEN_FALSE (\t. SND(I_srdy_E(e' t)))(t' + 1,t'rdy1)"
          ASSUME_TAC
   THENL [
     % Subgoal 2.1: (New subgoal) %
     REWRITE_ASSUM_TAC
       ("NTH_TIME_FALSE 1(\t. SND(I_srdy_E(e' t)))(ti' + 1)t'rdy1",
        [num_CONV "1";NTH_TIME_FALSE])
     THEN REWRITE_ASSUM_TAC
       ("NTH_TIME_FALSE 0(\t. SND(I_srdy_E(e' t)))(ti' + 1)t'",
        [NTH_TIME_FALSE])
     THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
     THEN RULE_ASSUM_TAC (REWRITE_RULE [ADD1;ADD_CLAUSES])
     THEN IMP_RES_TAC STABLE_TRUE_THEN_FALSE_UNIQUE
     THEN FILTER_ASM_REWRITE_TAC (\tm. tm = "t' = (t'':timeC)") []
     THEN FILTER_ASM_REWRITE_TAC (\tm. not (is_eq tm)) []
   ;
     % Subgoal 2.2: (Continue) %
     SUBGOAL_THEN
       "STABLE_FALSE(Sack_Sig_Is_TRUE s' e')(t'rdy1,t'rdy1)" ASSUME_TAC
     THENL [
        % Subgoal 2.2.1: (New Subgoal) %
        REWRITE_TAC [STABLE_FALSE]
        THEN BETA_TAC
        THEN ASSUME_TAC (SPEC "t'rdy1:timeC" LESS_EQ_REFL)
        THEN ASM_REWRITE_TAC[]
        THEN REPEAT STRIP_TAC
        THEN SUBGOAL_THEN
          "t'' = (t'rdy1:timeC)" (\thm. RULE_ASSUM_TAC (REWRITE_RULE [thm]))
        THENL [
           % Subgoal 2.2.1.1: (New Subgoal) %
           IMP_RES_TAC LESS_EQUAL_ANTISYM
        ;
           % Subgoal 2.2.1.2: (Continue) %
           IMP_RES_TAC
            (REWRITE_RULE [bsig;BSel] I_LAST_STABLE_HI_FROM_T'RDY0_TO_T'RDY1)
           THEN NRULE_ASSUM_TAC
             ("STABLE_HI(\t:timeC. SND(I_last_O(p' t)))(t' + 1,t'rdy1)",
              (BETA_RULE o (REWRITE_RULE [STABLE_HI])))
           THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
           THEN SPEC_ASSUM_TAC
                  ("!t.(t'+1)<=t /\ t<=t'rdy1 ==> (SND(I_last_O(p' t))=HI)",
                   "t'rdy1:timeC")
           THEN RES_TAC
           THEN IMP_RES_TAC I_LAST_HI_IMP_SACK_SIG_NOT_TRUE
        ]
     ;
        % Subgoal 2.2.2: (Continue) %
        SUBGOAL_THEN "1 <= t'rdy2" ASSUME_TAC
        THENL [
           % Subgoal 2.2.2.1: (New Subgoal) %
           IMP_RES_TAC (RIMP ONE_LESS_EQ)
           THEN REWRITE_ASSUM_TAC
                 ("NTH_TIME_FALSE 1(\t. SND(I_srdy_E(e' t)))(ti' + 1)t'rdy1",
                  [num_CONV "1";NTH_TIME_FALSE;STABLE_TRUE_THEN_FALSE])
           THEN REWRITE_ASSUM_TAC
                  ("STABLE_TRUE_THEN_FALSE(\t. SND(I_srdy_E(e' t)))
                    (t'rdy1+1,t'rdy2)",[STABLE_TRUE_THEN_FALSE])
           THEN POP_ASSUM_LIST
                  (MAP_EVERY (\thm. STRIP_ASSUME_TAC
                                                  (REWRITE_RULE [ADD1;ADD_CLAUSES] thm)))
                    THEN ASSUME_TAC (SPECL ["ti':timeC";"1"] LESS_EQ_ADD)
                    THEN ASSUME_TAC (SPECL ["t'':timeC";"1"] LESS_EQ_ADD)
                    THEN ASSUME_TAC (SPECL ["t'rdy1:timeC";"1"] LESS_EQ_ADD)
                    THEN IMP_RES_TAC LESS_EQ_TRANS
                    THEN IMP_RES_TAC LESS_EQ_TRANS
                 ;
                    % Subgoal 2.2.2.2: (Continue) %
                    IMP_RES_TAC I_SRDY_TRUE_IMP_SACK_SIG_NOT_TRUE
                    THEN IMP_RES_TAC
                           (REWRITE_RULE [bsig;BSel] SACK_SIG_FALSE_DURING_DATA_0_1)
                    THEN SUBGOAL_THEN
                           "(ti' <= (t'rdy2-1)) /\
                            (t'rdy1 <= (t'rdy1-1)+1)"
                                STRIP_ASSUME_TAC
                    THENL [
                      % Subgoal 2.2.2.2.1: (New subgoal) %
                      REWRITE_ASSUM_TAC
                        ("STABLE_FALSE(Sack_Sig_Is_TRUE s' e')(ti',t'rdy1 - 1)",
                         [STABLE_FALSE])
                      THEN REWRITE_ASSUM_TAC
                             ("STABLE_TRUE_THEN_FALSE (\t. SND(I_srdy_E(e' t)))
                              (t'rdy1 + 1,t'rdy2)",[STABLE_TRUE_THEN_FALSE])
                      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
                      THEN IMP_RES_TAC
                             (SPECL ["t'rdy1+1";"t'rdy2:timeC";"1"] LESS_EQ_MONO_SUB)
                      THEN REWRITE_ASSUM_TAC
                        ("((t'rdy1 + 1) - 1) <= (t'rdy2 - 1)",[ADD_SUB])
                      THEN IMP_RES_TAC (RIMP ONE_LESS_EQ)
                      THEN ASSUME_TAC (SPECL ["t'rdy1:timeC";"1"] SUB_LESS_EQ)
                      THEN IMP_RES_TAC LESS_EQ_TRANS
                      THEN IMP_RES_TAC (SPECL ["t'rdy1:timeC";"1"] SUB_ADD)
                      THEN IMP_RES_TAC
                             (SPECL ["ti'+1";"t'rdy1:timeC";"1"] LESS_EQ_MONO_SUB)
                      THEN REWRITE_ASSUM_TAC
                             ("((ti' + 1) - 1) <= (t'rdy1 - 1)",[ADD_SUB])
                      THEN IMP_RES_TAC LESS_EQ_TRANS
                      THEN ASSUME_TAC (SPEC "t'rdy1:timeC" LESS_EQ_REFL)
                      THEN ASM_REWRITE_TAC[]
                    ;
                      % Subgoal 2.2.2.2.2: (Continue) %
                      IMP_RES_TAC SUP_INTERVAL_STABLE_FALSE
                      THEN ASM_REWRITE_TAC[]
                    ]
                 ]
              ]
           ]
        ]
      ));

let P_DOWN_TRUE_THEN_STABLE_FALSE_FROM_T'RDY1_TO_T'RDY2 = TAC_PROOF
    (([],
     "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
        (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
        (tp' ti' t'rdy1 t'rdy2 :timeC) .
     (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
      NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti' + 1)t'rdy1 /\
      STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(t'rdy1 + 1,t'rdy2) /\
      (VAL 1(SUBARRAY(SND(L_ad_inE(e' tp')))(1,0))) > 1) ==>
         TRUE_THEN_STABLE_FALSE(\u'. P_downS(s' u'))(t'rdy1 + 1,t'rdy2)"),
     REWRITE_TAC [TRUE_THEN_STABLE_FALSE]
     THEN BETA_TAC
     THEN REPEAT STRIP_TAC
     THEN IMP_RES_TAC SACK_SIG_FALSE_DURING_DATA_1_2
     THENL [
        % Subgoal 1: "(t'rdy1 + 1) <= t'rdy2" %
        REWRITE_ASSUM_TAC
          ("STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(t'rdy1 + 1,t'rdy2)",
           [STABLE_TRUE_THEN_FALSE])
        THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
        THEN ASM_REWRITE_TAC[]
     ;
    % Subgoal 2: "P_downS(s'(t'rdy1 + 1))" %
    IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
    THEN IMP_RES_TAC NTH_TIME_TRUE_X_IMP_X
    THEN IMP_RES_TAC NEW_STATE_PD_FROM_TI'_TO_T'SACK_1
    THEN NRULE_ASSUM_TAC
          ("NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti' + 1)t'rdy1",
           (BETA_RULE o
            (REWRITE_RULE [num_CONV "1";NTH_TIME_FALSE;STABLE_TRUE_THEN_FALSE;
                           bsig;BSel])))
    THEN CHOOSE_ASSUM_TAC
           "?t. ((ti' + (SUC 0)) <= t /\
             (!t'. (ti' + (SUC 0)) <= t' /\ t' < t ==> SND(I_srdy_E(e' t'))) /\
             ~SND(I_srdy_E(e' t))) /\ (t + (SUC 0)) <= t'rdy1 /\
              (!t'. (t + (SUC 0)) <= t' /\ t'<t'rdy1 ==> SND(I_srdy_E(e' t')))
              /\ ~SND(I_srdy_E(e' t'rdy1))"
    THEN POP_ASSUM_LIST
          (MAP_EVERY
           (\thm. STRIP_ASSUME_TAC (REWRITE_RULE [ADD1;ADD_CLAUSES] thm)))
    THEN REWRITE_ASSUM_TAC
           ("STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(t'rdy1 + 1,t'rdy2)",
            [STABLE_TRUE_THEN_FALSE])
    THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
    THEN ASSUME_TAC (SPECL ["t':timeC";"1"] LESS_EQ_ADD)
    THEN ASSUME_TAC (SPECL ["t'rdy1:timeC";"1"] LESS_EQ_ADD)
    THEN IMP_RES_TAC LESS_EQ_TRANS
    THEN NRULE_ASSUM_TAC
           ("!t'. (ti'+1)<=t' ==> t'<=t'rdy2 ==> New_State_Is_PD s' e' t'",
            ((REWRITE_RULE [New_State_Is_PD]) o (SPEC "t'rdy1:timeC")))
    THEN RES_TAC
    THEN IMP_RES_TAC P_down_ISO
    THEN ASM_REWRITE_TAC[]
 ;
    % Subgoal 3:  [ "(t'rdy0 + 1) < t'" ]
                  [ "t' <= t'rdy1" ]
                  [ "P_downS(s' t')" ] %
    UNDISCH_TAC "P_downS(s' (t':timeC))"
    THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
    THEN SUBGOAL_THEN "t' = (t'-1)+1" (\thm. PURE_ONCE_REWRITE_TAC [thm])
    THENL [
       % Subgoal 3.1: (New Subgoal) %
       IMP_RES_TAC (RIMP ONE_LESS_EQ)
       THEN REWRITE_ASSUM_TAC
              ("NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti' + 1)t'rdy1",
               [num_CONV "1";NTH_TIME_FALSE;STABLE_TRUE_THEN_FALSE])
       THEN POP_ASSUM_LIST
              (MAP_EVERY
               (\thm. STRIP_ASSUME_TAC (REWRITE_RULE [ADD1;ADD_CLAUSES] thm)))
       THEN ASSUME_TAC (SPECL ["ti':timeC";"1"] LESS_EQ_ADD)
       THEN ASSUME_TAC (SPECL ["t'':timeC";"1"] LESS_EQ_ADD)
       THEN ASSUME_TAC (SPECL ["t'rdy1:timeC";"1"] LESS_EQ_ADD)
       THEN IMP_RES_TAC LT_IMP_LE
       THEN IMP_RES_TAC LESS_EQ_TRANS
       THEN IMP_RES_TAC LESS_EQ_TRANS
       THEN IMP_RES_TAC (SPECL ["t':timeC";"1"] (SYM_RULE SUB_ADD))
    ;
       % Subgoal 3.2: (Continue) %
       IMP_RES_TAC P_down_ISO
       THEN ASM_REWRITE_TAC[]
       THEN NRULE_ASSUM_TAC
             ("STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(t'rdy1 + 1,t'rdy2)",
              (BETA_RULE o (REWRITE_RULE [STABLE_TRUE_THEN_FALSE;bsig;BSel])))
       THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
       THEN SPEC_ASSUM_TAC
              ("!t. (t'rdy1 + 1) <= t /\ t < t'rdy2 ==> SND(I_srdy_E(e' t))",
               "t'-1")
       THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LT_IMP_LE_PRE)
       THEN SUBGOAL_THEN "1 <= t'" ASSUME_TAC
       THENL [
         % Subgoal 3.2.1: "1 <= t'" %
         IMP_RES_TAC (RIMP ONE_LESS_EQ)
         THEN ASSUME_TAC (SPECL ["ti':timeC";"1"] LESS_EQ_ADD)
         THEN REWRITE_ASSUM_TAC
```

```
              ("NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti' + 1)t'rdy1",
               [num_CONV "1";NTH_TIME_FALSE;STABLE_TRUE_THEN_FALSE])
        THEN POP_ASSUM_LIST
             (MAP_EVERY
              (\thm. STRIP_ASSUME_TAC (REWRITE_RULE [ADD1;ADD_CLAUSES] thm)))
        THEN ASSUME_TAC (SPECL ["t'':timeC";"1"] LESS_EQ_ADD)
        THEN ASSUME_TAC (SPECL ["t'rdy1:timeC";"1"] LESS_EQ_ADD)
        THEN IMP_RES_TAC LT_IMP_LE
        THEN IMP_RES_TAC LESS_EQ_TRANS
        THEN IMP_RES_TAC LESS_EQ_TRANS
       ;
        % Subgoal 3.2.2: (Continue) %
        IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LE_IMP_PRE_LT)
        THEN RES_TAC
        THEN ASM_REWRITE_TAC[]
       ]
     ]
   ]
  );;

let OFFSET_P_SIZE_STABLE_FROM_T'RDY1_TO_T'RDY2 = TAC_PROOF
  (([],
   "! (u' :timeC)
      (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
      (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
      (tp' ti' t'rdy1 t'rdy2 :timeC) .
    (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
     NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti' + 1)t'rdy1 /\
     STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (t'rdy1+1,t'rdy2) /\
     VAL 1 (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0)) > 1 /\
     ((t'rdy1+u'+2) <= t'rdy2+1))
     ==>
    (P_sizeS (s' (t'rdy1+u'+2)) =
       DECN 1 (DECN 1 (SUBARRAY (SND(L_ad_inE(e' tp'))) (1,0))))"),
   INDUCT_TAC
   THEN REWRITE_TAC [ADD1;ADD_CLAUSES;ADD_ASSOC;SYM_RULE (REDUCE_CONV "1+1")]
   THEN REPEAT STRIP_TAC
   THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
   THEN IMP_RES_TAC SACK_SIG_FALSE_DURING_DATA_1_2
   THEN IMP_RES_TAC P_LOAD_TRUE_THEN_STABLE_FALSE_FROM_TP'_TO_T'SACK
   THEN IMP_RES_TAC P_DOWN_TRUE_THEN_STABLE_FALSE_FROM_T'RDY1_TO_T'RDY2
   THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "0<1"))
   THEN IMP_RES_TAC PRIOR_FALSE_EVENTS_EXIST
   THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "1>0"))
   THEN IMP_RES_TAC GREATER_TRANS
   THENL [
      % Subgoal 1: (Base Case) %
      IMP_RES_TAC P_size_ISO
      THEN ONCE_ASM_REWRITE_TAC[]
      THEN POP_ASSUM (\thm. ALL_TAC) %KEEP%
      THEN SUBGOAL_THEN
           "tp' < t'rdy1 /\ (t'rdy1+1) <= t'rdy2" STRIP_ASSUME_TAC
      THENL [
         % Subgoal 1.1: (New Subgoal) %
         IMP_RES_TAC NTH_TRANS_CAUSAL
         THEN ASSUME_TAC (SPEC "ti':timeC" (REWRITE_RULE [ADD1] LESS_SUC_REFL))
         THEN REWRITE_ASSUM_TAC
              ("NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti' + 1)t'rdy1",
               [num_CONV "1";NTH_TIME_FALSE;STABLE_TRUE_THEN_FALSE])
         THEN REWRITE_ASSUM_TAC
              ("STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(t'rdy1 + 1,t'rdy2)",
               [STABLE_TRUE_THEN_FALSE])
         THEN POP_ASSUM_LIST
              (MAP_EVERY
               (\thm. STRIP_ASSUME_TAC (REWRITE_RULE [ADD1;ADD_CLAUSES] thm)))
         THEN ASSUME_TAC (SPECL ["t'':timeC";"1"] LESS_EQ_ADD)
         THEN IMP_RES_TAC LESS_EQ_TRANS
         THEN IMP_RES_TAC LESS_EQ_LESS_TRANS
         THEN IMP_RES_TAC LESS_LESS_EQ_TRANS
         THEN IMP_RES_TAC LESS_EQ_TRANS
         THEN IMP_RES_TAC LT_IMP_LE
         THEN ASM_REWRITE_TAC[]
      ;
        % Subgoal 1.2: (Continue) %
        ASSUME_TAC (SPECL ["t'rdy1:timeC";"1"] LESS_EQ_ADD)
        THEN IMP_RES_TAC LESS_LESS_EQ_TRANS
        THEN IMP_RES_TAC LESS_EQ_TRANS
        THEN NRULE_ASSUM_TAC
               ("TRUE_THEN_STABLE_FALSE(\u'. P_loadS(s' u'))(tp',t'rdy2)",
                (BETA_RULE o (REWRITE_RULE [TRUE_THEN_STABLE_FALSE])))
        THEN NRULE_ASSUM_TAC
               ("TRUE_THEN_STABLE_FALSE(\u'. P_downS(s' u'))(t'rdy1+1,t'rdy2)",
                (BETA_RULE o (REWRITE_RULE [TRUE_THEN_STABLE_FALSE])))
        THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
        THEN SPEC_ASSUM_TAC
               ("!t. tp' < t /\ t <= t'rdy2 ==> ~P_loadS(s' t)","t'rdy1+1")
        THEN RES_TAC
        THEN ASM_REWRITE_TAC[]
        THEN NRULE_ASSUM_TAC
               ("NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti' + 1)t'rdy1",
                (PURE_ONCE_REWRITE_RULE [num_CONV "1"]))
        THEN NRULE_ASSUM_TAC
               ("NTH_TIME_FALSE (SUC 0)(bsig I_srdy_E e')(ti'+(SUC 0))t'rdy1",
                (PURE_ONCE_REWRITE_RULE [NTH_TIME_FALSE]))
        THEN CHOOSE_ASSUM_TAC
               "?t. NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + (SUC 0))t /\
                STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(t + 1,t'rdy1)"
        THEN POP_ASSUM_LIST
               (MAP_EVERY
                (\thm. STRIP_ASSUME_TAC (REWRITE_RULE [ADD1;ADD_CLAUSES] thm)))
        THEN SUBGOAL_THEN
               "((t''+2) <= t'rdy1+1) /\ ((t'rdy1+1) <= t'rdy1+1)"
               STRIP_ASSUME_TAC
        THENL [
          % Subgoal 1.2.1: (New subgoal) %
          ASSUME_TAC (SPEC "t'rdy1+1" LESS_EQ_REFL)
          THEN REWRITE_ASSUM_TAC
                 ("STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(t'' + 1,t'rdy1)",
                  [STABLE_TRUE_THEN_FALSE])
          THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
          THEN IMP_RES_TAC
                 (SPECL ["t''+1";"t'rdy1:timeC";"1"] (RIMP LESS_EQ_MONO_ADD_EQ))
          THEN ASM_REWRITE_TAC [SYM_RULE (REDUCE_CONV "1+1");ADD_ASSOC]
        ;
          % Subgoal 1.2.2: (Continue) %
          IMP_RES_TAC (SPEC "t'rdy+1" P_SIZE_STABLE_FROM_T'RDY0_TO_T'RDY1)
          THEN ASM_REWRITE_TAC[]
        ]
      ]
   ;
      % Subgoal 2: (Induction Step) %
      IMP_RES_TAC P_size_ISO
      THEN ONCE_ASM_REWRITE_TAC[]
      THEN POP_ASSUM (\thm. ALL_TAC) %KEEP%
      THEN RULE_ASSUM_TAC
             (\thm. REWRITE_RULE [SYM_RULE (REDUCE_CONV "1+1");ADD_ASSOC] thm)
      THEN ASSUME_TAC (SPECL ["(((t'rdy1 + u') + 1) + 1)";"1"] LESS_EQ_ADD)
      THEN IMP_RES_TAC LESS_EQ_TRANS
      THEN RES_TAC
      THEN NRULE_ASSUM_TAC
             ("TRUE_THEN_STABLE_FALSE(\u'. P_loadS(s' u'))(tp',t'rdy2)",
              (BETA_RULE o (REWRITE_RULE [TRUE_THEN_STABLE_FALSE])))
      THEN NRULE_ASSUM_TAC
             ("TRUE_THEN_STABLE_FALSE(\u'. P_downS(s' u'))(t'rdy1 + 1,t'rdy2)",
              (BETA_RULE o (REWRITE_RULE [TRUE_THEN_STABLE_FALSE])))
      THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
      THEN SPEC_ASSUM_TAC
             ("!t. tp' < t /\ t <= t'rdy2 ==> ~P_loadS(s' t)",
              "((t'rdy1+u')+1)+1")
      THEN SPEC_ASSUM_TAC
             ("!t. (t'rdy1 + 1) < t /\ t <= t'rdy2 ==> ~P_downS(s' t)",
              "((t'rdy1+u')+1)+1")

      THEN IMP_RES_TAC
```

```
              (SPECL ["(((t'rdyl+u')+1)+1)+1";"t'rdy2+1";"1"] LESS_EQ_MONO_SUB)
       THEN REWRITE_ASSUM_TAC
              ("(((((t'rdy1 + u') + 1) + 1) + 1) - 1) <= ((t'rdy2 + 1) - 1)",
               [ADD_SUB])
       THEN ASSUME_TAC (SPECL ["t'rdy1:timeC";"u':timeC"] LESS_EQ_ADD)
       THEN IMP_RES_TAC
              (RIMP (SPECL ["t'rdy1:timeC";"t'rdy1+u'";"1"] LESS_EQ_MONO_ADD_EQ))
       THEN ASSUME_TAC (SPEC "(t'rdy1+u')+1" (REWRITE_RULE [ADD1] LESS_SUC_REFL))
       THEN IMP_RES_TAC LESS_EQ_LESS_TRANS

       THEN IMP_RES_TAC NTH_TRANS_CAUSAL
       THEN ASSUME_TAC (SPECL ["ti':timeC";"1"] LESS_EQ_ADD)
       THEN NRULE_ASSUM_TAC
              ("NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti' + 1)t'rdy1",
               (BETA_RULE o
                (REWRITE_RULE [num_CONV "1";NTH_TIME_FALSE;
                               STABLE_TRUE_THEN_FALSE])))
       THEN POP_ASSUM_LIST
              (MAP_EVERY
               (\thm. STRIP_ASSUME_TAC (REWRITE_RULE [ADD1;ADD_CLAUSES] thm)))
       THEN ASSUME_TAC (SPEC "t'':timeC" (REWRITE_RULE [ADD1] LESS_SUC_REFL))
       THEN ASSUME_TAC (SPECL ["t'rdy1:timeC";"1"] LESS_EQ_ADD)

       THEN IMP_RES_TAC LESS_EQ_TRANS
       THEN IMP_RES_TAC LESS_EQ_LESS_TRANS
       THEN IMP_RES_TAC LESS_LESS_EQ_TRANS
       THEN IMP_RES_TAC LESS_TRANS
       THEN RES_TAC
       THEN ASM_REWRITE_TAC[]
    ]
   );;

let P_SIZE_STABLE_FROM_T'RDY1_TO_T'RDY2 = TAC_PROOF
   (([],
    "! (t' :timeC)
       (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' t'rdy1 t'rdy2 :timeC) .
     (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
      NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti' + 1)t'rdy1 /\
      STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (t'rdy1+1,t'rdy2) /\
      VAL 1 (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0)) > 1 /\
      ((t'rdy1+2) <= t') /\
      (t' <= t'rdy2+1))
      ==>
     (P_sizeS (s' t') =
       DECN 1 (DECN 1 (SUBARRAY (SND(L_ad_inE (e' tp'))) (1,0))))"),
    REWRITE_TAC [SYM_RULE (REDUCE_CONV "1+1");ADD_ASSOC]
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC (SPEC "t'-tp'+2" OFFSET_P_SIZE_STABLE_FROM_T'RDY1_TO_T'RDY2)
    THEN SPECL_ASSUM_TAC
           ("!(t' tp''':timeC). (t'rdy1+((t'-(tp'''+2))+2)) <= (t'rdy2 + 1) ==>
              (P_sizeS(s'(t'rdy1 + ((t' - (tp''' + 2)) + 2))) =
              DECN 1(DECN 1(SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0))))",
             ["t':timeC";"t'rdy1:timeC"])
    THEN SUBGOAL_THEN
            "(t'rdy1 + ((t' - (t'rdy1 + 2)) + 2)) = t'"
            (\thm. RULE_ASSUM_TAC (REWRITE_RULE [thm]))
     THENL [
        % Subgoal 1: -New subgoal- "t'rdy1 + ((t' - (t'rdy1 + 2)) + 2) = t'" %
        REWRITE_TAC [SYM_RULE (ASSOC_SUB_SUB1)]
        THEN SUBGOAL_THEN "2 <= (t' - t'rdy1)" ASSUME_TAC
        THENL [
           % Subgoal 1.1: "2 <= (t' - t'rdy1)"
                            [ "((t'rdy1 + 1) + 1) <= t'" ] %
           REWRITE_TAC
             [SYM_RULE (SPECL ["2";"t'-t'rdy1";"t'rdy1:timeC"]
                              LESS_EQ_MONO_ADD_EQ)]
           THEN ASSUME_TAC (SPECL ["t'rdy1:timeC";"1"] LESS_EQ_ADD)
           THEN ASSUME_TAC (SPECL ["t'rdy1+1";"1"] LESS_EQ_ADD)
           THEN IMP_RES_TAC LESS_EQ_TRANS
           THEN IMP_RES_TAC (SPECL ["t':timeC";"t'rdy1:timeC"] SUB_ADD)
```

```
              THEN ASM_REWRITE_TAC[]
              THEN PURE_ONCE_REWRITE_TAC [ADD_SYM]
              THEN ASM_REWRITE_TAC[SYM_RULE (REDUCE_CONV "1+1");ADD_ASSOC]
           ;
              % Subgoal 1.2: [ "2 <= (t' - t'rdy1)" ] %
              ASSUME_TAC (SPEC "2" LESS_EQ_REFL)
              THEN IMP_RES_TAC (SPECL ["t'-t'rdy1";"2";"2"] ASSOC_SUB_ADD1)
              THEN ASM_REWRITE_TAC[SUB_EQUAL_0;ADD_CLAUSES]
              THEN PURE_ONCE_REWRITE_TAC [ADD_SYM]
              THEN ASSUME_TAC (SPECL ["t'rdy1:timeC";"1"] LESS_EQ_ADD)
              THEN ASSUME_TAC (SPECL ["t'rdy1+1";"1"] LESS_EQ_ADD)
              THEN IMP_RES_TAC LESS_EQ_TRANS
              THEN IMP_RES_TAC (SPECL ["t':timeC";"t'rdy1:timeC"] SUB_ADD)
           ]
       ;
          % Subgoal 2 %
          RES_TAC
       ]
     );;

let DECN_DECN_WORDN_1_NOT_EQ = mk_thm
    ([],
     "! (x :wordn) (m n :num) .
      (VAL 1 x > n) ==>
       (n > m + 1) ==>
        ~(DECN 1 (DECN 1 x) = WORDN 1 m)"
    );;

let I_LAST_STABLE_HI_FROM_T'RDY1_TO_T'RDY2 = TAC_PROOF
    (([],
     "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
        (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
        (tp' ti' t'rdy1 t'rdy2 :timeC) .
      (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
       NTH_TIME_FALSE 1 (bsig I_srdy_E e') (ti'+1) t'rdy1 /\
       STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (t'rdy1+1,t'rdy2) /\
       (VAL 1 (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0)) > 2)) ==>
          STABLE_HI (bsig I_last_O p') (t'rdy1+1,t'rdy2)"),
     REWRITE_TAC [bsig;BSel;STABLE_HI]
     THEN REPEAT STRIP_TAC
     THEN IMP_RES_TAC (REWRITE_RULE [bsig;BSel] IB_READY_ASSUMPS)
     THENL [
        % Subgoal 1: "(t'rdy1 + 1) <= t'rdy2" %
        REWRITE_ASSUM_TAC
           ("STABLE_TRUE_THEN_FALSE(\t. SND(I_srdy_E(e' t)))(t'rdy1 + 1,t'rdy2)",
            [STABLE_TRUE_THEN_FALSE])
        THEN ASM_REWRITE_TAC[]
       ;
        % Subgoal 2:
          "(\t. SND(I_last_O(p' t)))t' = HI"
          [ "STABLE_TRUE_THEN_FALSE(\t. SND(I_srdy_E(e' t)))(t'rdy1 + 1,t'rdy2)" ]
          [ "(t'rdy1 + 1) <= t'" ]
          [ "t' <= t'rdy2" ] %
        BETA_TAC
        THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
        THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "2>1"))
        THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "1>0"))
        THEN IMP_RES_TAC GREATER_TRANS
        THEN SUBGOAL_THEN "(tp'<=ti') /\ ((ti'+1) <= t'rdy1)" STRIP_ASSUME_TAC
        THENL [
          % Subgoal 2.1: (New subgoal) %
          IMP_RES_TAC NTH_TRANS_CAUSAL
          THEN REWRITE_ASSUM_TAC
                 ("NTH_TIME_FALSE 1(\t. SND(I_srdy_E(e' t)))(ti' + 1)t'rdy1",
                  [num_CONV "1";NTH_TIME_FALSE;STABLE_TRUE_THEN_FALSE])
          THEN POP_ASSUM_LIST
                 (MAP_EVERY
                  (\thm. STRIP_ASSUME_TAC (REWRITE_RULE [ADD1;ADD_CLAUSES] thm)))
          THEN ASSUME_TAC (SPECL ["t'':timeC";"1"] LESS_EQ_ADD)
          THEN IMP_RES_TAC LESS_EQ_TRANS
          THEN ASM_REWRITE_TAC[]
        ;
```

```
% Subgoal 2.2: (Continue) %
ASSUME_TAC (SPECL ["t'rdy1:timeC";"1"] LESS_EQ_ADD)
THEN ASSUME_TAC (SPECL ["t'rdy2:timeC";"1"] LESS_EQ_ADD)
THEN IMP_RES_TAC LESS_EQ_TRANS
THEN IMP_RES_TAC
       (REWRITE_RULE [bsig;BSel] SACK_SIG_FALSE_DURING_DATA_1_2)
THEN IMP_RES_TAC NTH_TIME_TRUE_X_IMP_X
THEN IMP_RES_TAC NEW_STATE_PD_FROM_TI'_TO_T'SACK_1
THEN IMP_RES_TAC
       (REWRITE_RULE
          [bsig;BSel]
          P_DOWN_TRUE_THEN_STABLE_FALSE_FROM_T'RDY1_TO_T'RDY2)
THEN NRULE_ASSUM_TAC
      ("TRUE_THEN_STABLE_FALSE(\u'.P_downS(s' u'))(t'rdy1+1,t'rdy2)",
       (BETA_RULE o (REWRITE_RULE [TRUE_THEN_STABLE_FALSE])))
THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
THEN SPEC_ASSUM_TAC
       ("!t. (t'rdy1 + 1) < t /\ t <= t'rdy2 ==> ~P_downS(s' t)",
        "t':timeC")
THEN REWRITE_ASSUM_TAC ("New_State_Is_PD s' e' t'",[New_State_Is_PD])
THEN IMP_RES_TAC I_last_ISO
THEN ASM_REWRITE_TAC
       [WIRE;SYM_RULE (prove_constructors_distinct pfsm_ty_Axiom)]
THEN POP_ASSUM (\thm. ALL_TAC)
THEN ASM_CASES_TAC "(t'rdy1+1) < t'"
THENL [
  % Subgoal 2.2.1: [ "(t'rdy1 + 1) < t'" ] %
  IMP_RES_TAC (REWRITE_RULE [ADD1] LT_IMP_SUC_LE)
  THEN NRULE_ASSUM_TAC
         ("((t'rdy1 + 1) + 1) <= t'",
          (REDUCE_RULE o (REWRITE_RULE [SYM_RULE ADD_ASSOC])))
  THEN IMP_RES_TAC
       (REWRITE_RULE [bsig;BSel] P_SIZE_STABLE_FROM_T'RDY1_TO_T'RDY2)
  THEN RES_TAC
  THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "2>0+1"))
  THEN IMP_RES_TAC
         (SPECL ["SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0)";
                 "0";"2"] DECN_DECN_WORDN_1_NOT_EQ)
  THEN ASM_REWRITE_TAC[]
;
  % Subgoal 2.2.2: [ "~(t'rdy1 + 1) < t'" ] %
  SUBGOAL_THEN "t' = t'rdy1+1" (\thm. REWRITE_TAC [thm])
  THENL [
    % Subgoal 2.2.2.1: (New Subgoal) %
    IMP_RES_TAC NOT_LESS
    THEN IMP_RES_TAC LESS_EQUAL_ANTISYM
  ;
    % Subgoal 2.2.2.2: (Continue) %
    ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "0<1"))
    THEN IMP_RES_TAC PRIOR_FALSE_EVENTS_EXIST
    THEN NRULE_ASSUM_TAC
         ("NTH_TIME_FALSE 1(\t. SND(I_srdy_E(e' t)))(ti' + 1)t'rdy1",
          (PURE_ONCE_REWRITE_RULE [num_CONV "1"]))
    THEN NRULE_ASSUM_TAC
         ("NTH_TIME_FALSE (SUC 0)(\t. SND(I_srdy_E(e' t)))
                          (ti' + (SUC 0))t'rdy1",
          (PURE_ONCE_REWRITE_RULE [NTH_TIME_FALSE]))
    THEN POP_ASSUM_LIST
          (MAP_EVERY
           (\thm. STRIP_ASSUME_TAC
                  (REWRITE_RULE [ADD1;ADD_CLAUSES] thm)))
    THEN SUBGOAL_THEN
          "((t'''+2) <= t'rdy1+1) /\
           ((t'rdy1 +1) <= t'rdy1 +1)" STRIP_ASSUME_TAC
    THENL [
      % Subgoal 2.2.2.2.1: (New subgoal) %
      ASSUME_TAC (SPEC "t'rdy1+1" LESS_EQ_REFL)
      THEN IMP_RES_TAC (REWRITE_RULE [ADD1] LT_IMP_SUC_LE)
      THEN IMP_RES_TAC
             (SPECL ["t''+1";"t'rdy1:timeC";"1"]
                    (RIMP LESS_EQ_MONO_ADD_EQ))
      THEN REWRITE_ASSUM_TAC
```

```
                        ("((t'' + 1) + 1) <= (t'rdy1 + 1)",
                         [ASSOC_ADD_ADD1;REDUCE_CONV "1+1"])
                THEN IMP_RES_TAC FALSE_EVENT_TIMES_EQUAL
                THEN PURE_ONCE_ASM_REWRITE_TAC[]
                THEN FILTER_ASM_REWRITE_TAC (\tm. not (is_eq tm)) []
              ;
                % Subgoal 2.2.2.2.2: (Continue) %
                IMP_RES_TAC
                  (REWRITE_RULE [bsig;BSel]
                               P_SIZE_STABLE_FROM_T'RDY0_TO_T'RDY1)
                THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "2>1"))
                THEN IMP_RES_TAC
                        (SPECL ["SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0)";
                                "1";"2"] DECN_WORDN_1_NOT_EQ)
                THEN ASM_REWRITE_TAC[]
              ]
            ]
          ]
        ]
      ]
    );;

let I_LAST_FOR_BLOCK_SIZE_0' = TAC_PROOF
   (([],
    "! (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' :timeC) .
     (Standard_Assumps PT_Write s e p t s' e' p' tp' ti' /\
      (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0) = WORDN 1 0)) ==>
        STABLE_LO
          (bsig I_last_O p')
          (ti' + 1,(@u'. NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)u'))"),
    REWRITE_TAC [STABLE_LO;bsig;BSel;NTH_TIME_FALSE]
    THEN REPEAT STRIP_TAC
    THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
    THENL [
       % Subgoal 1:
          "(ti' + 1) <=
           (@u'. STABLE_TRUE_THEN_FALSE(\t. SND(I_srdy_E(e' t)))(ti' + 1,u'))" %
       SUBGOAL_THEN
        "!t'rdy0.
          STABLE_TRUE_THEN_FALSE (\t. SND(I_srdy_E(e' t))) (ti'+1,t'rdy0) ==>
         ((@u'. STABLE_TRUE_THEN_FALSE (\t. SND(I_srdy_E(e' t)))(ti'+1,u'))
           = t'rdy0)"
        ASSUME_TAC
       THENL [
          % Subgoal 1.1: %
          REPEAT STRIP_TAC
          THEN SELECT_UNIQUE_TAC
          THEN ASM_REWRITE_TAC[]
          THEN REPEAT STRIP_TAC
          THEN IMP_RES_TAC STABLE_TRUE_THEN_FALSE_UNIQUE
       ;
          % Subgoal 1.2: %
          IMP_RES_TAC (REWRITE_RULE [bsig;BSel] IB_READY_ASSUMPS)
          THEN RES_TAC
          THEN REWRITE_ASSUM_TAC
                 ("STABLE_TRUE_THEN_FALSE(\t. SND(I_srdy_E(e' t)))(ti' + 1,u')",
                  [STABLE_TRUE_THEN_FALSE])
          THEN ASM_REWRITE_TAC[]
       ]
    ;
       % Subgoal 2:
        "(\t. SND(I_last_O(p' t)))t' = LO"
        [ "(ti' + 1) <= t'" ]
        [ "t' <= (@u'. NTH_TIME_FALSE 0(\t. SND(I_srdy_E(e' t)))(ti' + 1)u')" ] %
       BETA_TAC
       THEN SUBGOAL_THEN
        "!t'rdy0.
         NTH_TIME_FALSE 0 (\t. SND(I_srdy_E(e' t))) (ti' + 1) t'rdy0 ==>
        ((@u'. NTH_TIME_FALSE 0 (\t. SND(I_srdy_E(e' t)))(ti' + 1)u') = t'rdy0)"        ASSUME_TAC
       THENL [
          % Subgoal 2.1: %
          REPEAT STRIP_TAC
          THEN SELECT_UNIQUE_TAC
          THEN ASM_REWRITE_TAC[]
          THEN REPEAT STRIP_TAC
          THEN IMP_RES_TAC FALSE_EVENT_TIMES_EQUAL
        ;
          % Subgoal 2.2:
          "SND(I_last_O(p' t')) = LO"
              [ "(ti' + 1) <= t'" ]
              [ "t' <= (@u'.NTH_TIME_FALSE 0(\t.SND(I_srdy_E(e' t)))(ti'+1)u')" ]
              [ "!t'rdy0.
                  NTH_TIME_FALSE 0(\t. SND(I_srdy_E(e' t)))(ti' + 1)t'rdy0 ==>
                  ((@u'. NTH_TIME_FALSE 0(\t. SND(I_srdy_E(e' t)))(ti' + 1)u') =
                   t'rdy0)" ] %
          REWRITE_ASSUM_TAC
            ("!t'rdy0. NTH_TIME_FALSE 0(\t. SND(I_srdy_E(e' t)))(ti'+1)t'rdy0 ==>
              ((@u'. NTH_TIME_FALSE 0(\t. SND(I_srdy_E(e' t)))(ti' + 1)u') =
              t'rdy0)",[NTH_TIME_FALSE])
          THEN IMP_RES_TAC (REWRITE_RULE [bsig;BSel] IB_READY_ASSUMPS)
          THEN RES_TAC
          THEN ASM_REWRITE_ASSUM_TAC
                 ("t' <= (@u'. STABLE_TRUE_THEN_FALSE(\t. SND(I_srdy_E(e' t)))
                                (ti' + 1,u'))",[])
          THEN IMP_RES_TAC (REWRITE_RULE [bsig;BSel] I_LAST_FOR_BLOCK_SIZE_0)
          THEN NRULE_ASSUM_TAC
                 ("STABLE_LO(\t. SND(I_last_O(p' t)))(ti' + 1,u')",
                  (BETA_RULE o (REWRITE_RULE [STABLE_LO])))
          THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
          THEN SPEC_ASSUM_TAC
                 ("!t. (ti' + 1) <= t /\ t <= u' ==> (SND(I_last_O(p' t)) = LO)",
                  "t':timeC")
          THEN RES_TAC
        ]
     ]
    );;

let I_SRDY_FALSE_2_TIMES = TAC_PROOF
    (([],
     "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
        (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
        (tp' ti' :timeC) .
      (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
       VAL 1 (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0)) > 0) ==>
          ?t'rdy1. NTH_TIME_FALSE 1 (bsig I_srdy_E e') (ti'+1) t'rdy1"),
     REPEAT STRIP_TAC
     THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
     THEN IMP_RES_TAC IB_READY_ASSUMPS
     THEN SUBGOAL_THEN
            "?v'. STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(u' + 1,v')" ASSUME_TAC
     THENL [
        % Subgoal 1: (New Subgoal) %
        NRULE_ASSUM_TAC
          ("!u'. rdy_sig_ib e' p' u' ==>
            (?v'. STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(u' + 1,v'))",
           (BETA_RULE o (REWRITE_RULE [rdy_sig_ib;BSel]) o (SPEC "u':timeC")))
        THEN IMP_RES_TAC I_LAST_STABLE_HI_FROM_TI'_TO_T'RDY0
        THEN NRULE_ASSUM_TAC
               ("STABLE_HI(bsig I_last_O p')(ti' + 1,u')",
                (BETA_RULE o (REWRITE_RULE [STABLE_HI;bsig;BSel])))
        THEN NRULE_ASSUM_TAC
               ("STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(ti' + 1,u')",
                (BETA_RULE o (REWRITE_RULE [STABLE_TRUE_THEN_FALSE;bsig;BSel])))
        THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
        THEN SPEC_ASSUM_TAC
               ("!t. (ti' + 1) <= t /\ t <= u' ==> (SND(I_last_O(p' t)) = HI)",
                "u':timeC")
        THEN ASSUME_TAC (SPEC "u':timeC" LESS_EQ_REFL)
        THEN RES_TAC
        THEN RES_TAC
        THEN EXISTS_TAC "v':timeC"
        THEN ASM_REWRITE_TAC[]
```

```
   ;
      CHOOSE_ASSUM_TAC
        "?v'. STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(u' + 1,v')"
      THEN EXISTS_TAC "v':timeC"
      THEN REWRITE_TAC [num_CONV "1";NTH_TIME_FALSE]
      THEN EXISTS_TAC "u':timeC"
      THEN ASM_REWRITE_TAC [ADD1;ADD_CLAUSES]
   ]
  );;

let I_LAST_FOR_BLOCK_SIZE_1' = TAC_PROOF
   (([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' :timeC) .
     (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
      (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0) = WORDN 1 1)) ==>
        (~STABLE_LO (bsig I_last_O p')
                    (ti'+1,(@u'.NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti'+1)u')) /\
         STABLE_HI (bsig I_last_O p')
                   (ti'+1,(@u'.NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti'+1)u')) /\
         STABLE_LO (bsig I_last_O p')
                   ((@u'. NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)u')+1,
                    (@u'. NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti' + 1)u')))"),
    REPEAT GEN_TAC
    THEN STRIP_TAC
    THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
    THEN SUBGOAL_THEN
          "VAL 1 (SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0)) > 0" ASSUME_TAC
    THENL [
       % Subgoal 1: (New subgoal) %
       ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "1<=3"))
       THEN IMP_RES_TAC VAL_WORDN_IDENT_1
       THEN ASM_REWRITE_TAC[]
       THEN REDUCE_TAC
    ;
       % Subgoal 2: (Continue) %
       IMP_RES_TAC IB_READY_ASSUMPS
       THEN IMP_RES_TAC I_LAST_STABLE_HI_FROM_TI'_TO_T'RDY0
       THEN SUBGOAL_THEN
              "(@u'. NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)u') = u'"
              ASSUME_TAC
       THENL [
          % Subgoal 2.1: (New Subgoal) %
          SELECT_UNIQUE_TAC
          THENL [
            % Subgoal 2.1.1: (Existence) %
            REWRITE_TAC [NTH_TIME_FALSE]
            THEN ASM_REWRITE_TAC[]
          ;
            % Subgoal 2.1.2: (Uniqueness) %
            REPEAT STRIP_TAC
            THEN IMP_RES_TAC FALSE_EVENT_TIMES_EQUAL
          ]
       ;
          % Subgoal 2.2: (Continue) %
          IMP_RES_TAC STABLE_HI_IMP_NOT_STABLE_LO
          THEN ASM_REWRITE_TAC[]
          THEN IMP_RES_TAC I_SRDY_FALSE_2_TIMES
          THEN SUBGOAL_THEN
                 "NTH_TIME_FALSE 0 (bsig I_srdy_E e') (ti'+1) u' /\
                  STABLE_TRUE_THEN_FALSE (bsig I_srdy_E e') (u'+1,t'rdy1)"
                  STRIP_ASSUME_TAC
          THENL [
            % Subgoal 2.2.1: (New Subgoal) %
            REWRITE_ASSUM_TAC
              ("NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti' + 1)t'rdy1",
               [num_CONV "1";NTH_TIME_FALSE])
            THEN CHOOSE_ASSUM_TAC
                 "?t. STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(ti'+(SUC 0),t)/\
                      STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(t+(SUC 0),t'rdy1)"
            THEN POP_ASSUM_LIST
```

```
                (MAP_EVERY
                 (\thm. STRIP_ASSUME_TAC
                         (REWRITE_RULE [ADD1;ADD_CLAUSES] thm)))
          THEN SUBGOAL_THEN "u' = (t':timeC)" (\thm. REWRITE_TAC [thm])
          THENL [
            % Subgoal 2.2.1.1: (New subgoal) %
            IMP_RES_TAC STABLE_TRUE_THEN_FALSE_UNIQUE
          ;
            % Subgoal 2.2.1.2: (Continue) %
            ASM_REWRITE_TAC [NTH_TIME_FALSE]
          ]
        ;
          % Subgoal 2.2.2: (Continue) %
          IMP_RES_TAC I_LAST_FOR_BLOCK_SIZE_1
          THEN SUBGOAL_THEN
                "(@u'. NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti' + 1)u') = t'rdy1"
                ASSUME_TAC
          THENL [
            % Subgoal 2.2.2.1: (New Subgoal) %
            SELECT_UNIQUE_TAC
            THENL [
              % Subgoal 2.2.2.1.1: (Existence) %
              ASM_REWRITE_TAC[]
            ;
              % Subgoal 2.2.2.1.2: (Uniqueness) %
              REPEAT STRIP_TAC
              THEN IMP_RES_TAC FALSE_EVENT_TIMES_EQUAL
            ]
          ;
            % Subgoal 2.2.2.2: (Continue) %
            ASM_REWRITE_TAC[]
          ]
        ]
      ]
    ]
   );;

let I_SRDY_FALSE_3_TIMES = TAC_PROOF
   (([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' :timeC) .
     (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
      VAL 1 (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0)) > 1) ==>
         ?t'rdy2. NTH_TIME_FALSE 2 (bsig I_srdy_E e') (ti'+1) t'rdy2"),
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC EXPAND_STANDARD_ASSUMPS
    THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "1>0"))
    THEN IMP_RES_TAC GREATER_TRANS
    THEN IMP_RES_TAC I_SRDY_FALSE_2_TIMES
    THEN IMP_RES_TAC IB_READY_ASSUMPS
    THEN SUBGOAL_THEN
          "?v'. STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(t'rdy1 + 1,v')"
          ASSUME_TAC
    THENL [
       % Subgoal 1: (New Subgoal) %
       NRULE_ASSUM_TAC
         ("!u'. rdy_sig_ib e' p' u' ==>
           (?v'. STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(u' + 1,v'))",
          (BETA_RULE o (REWRITE_RULE [rdy_sig_ib;BSel]) o (SPEC "t'rdy1:timeC")))
       THEN ASSUME_TAC (REWRITE_RULE [] (REDUCE_CONV "0<1"))
       THEN IMP_RES_TAC PRIOR_FALSE_EVENTS_EXIST
       THEN NRULE_ASSUM_TAC
              ("NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti' + 1)t'rdy1",
               (PURE_ONCE_REWRITE_RULE [num_CONV "1"]))
       THEN NRULE_ASSUM_TAC
              ("NTH_TIME_FALSE (SUC 0)(bsig I_srdy_E e')(ti' + (SUC 0))t'rdy1",
               (PURE_ONCE_REWRITE_RULE [NTH_TIME_FALSE]))

       THEN IMP_RES_TAC I_LAST_STABLE_HI_FROM_T'RDY0_TO_T'RDY1
       THEN NRULE_ASSUM_TAC
              ("STABLE_HI(bsig I_last_O p')(ti' + 1,u')",
                (BETA_RULE o (REWRITE_RULE [STABLE_HI;bsig;BSel])))
        THEN NRULE_ASSUM_TAC
               ("STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(ti' + 1,u')",
                (BETA_RULE o (REWRITE_RULE [STABLE_TRUE_THEN_FALSE;bsig;BSel])))
        THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
        THEN SPEC_ASSUM_TAC
               ("!t. (ti' + 1) <= t /\ t <= u' ==> (SND(I_last_O(p' t)) = HI)",
                "u':timeC")
        THEN ASSUME_TAC (SPEC "u':timeC" LESS_EQ_REFL)
        THEN RES_TAC
        THEN RES_TAC
        THEN EXISTS_TAC "v':timeC"
        THEN ASM_REWRITE_TAC[]
     ;
        CHOOSE_ASSUM_TAC
          "?v'. STABLE_TRUE_THEN_FALSE(bsig I_srdy_E e')(u' + 1,v')"
        THEN EXISTS_TAC "v':timeC"
        THEN REWRITE_TAC [num_CONV "1";NTH_TIME_FALSE]
        THEN EXISTS_TAC "u':timeC"
        THEN ASM_REWRITE_TAC [ADD1;ADD_CLAUSES]
     ]
   );;

let I_LAST_FOR_BLOCK_SIZE_2' = mk_thm
   ([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' :timeC) .
     (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
      (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0) = WORDN 1 2)) ==>
        (~STABLE_LO (bsig I_last_O p')
                    (ti'+1,(@u'.NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti'+1)u')) /\
         ~STABLE_LO (bsig I_last_O p')
                    ((@u'. NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)u') + 1,
                     (@u'. NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti' + 1)u')) /\
         STABLE_HI (bsig I_last_O p')
                   (ti'+1,(@u'.NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti'+1)u')) /\
         STABLE_LO (bsig I_last_O p')
                   ((@u'. NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti' + 1)u')+1,
                    (@u'. NTH_TIME_FALSE 2(bsig I_srdy_E e')(ti' + 1)u')))"
   );;

let I_LAST_FOR_BLOCK_SIZE_3' = mk_thm
   ([],
    "! (pti :PTI) (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (t :timeT) (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out)
       (tp' ti' :timeC) .
     (Standard_Assumps pti s e p t s' e' p' tp' ti' /\
      (SUBARRAY(SND(L_ad_inE(e' tp')))(1,0) = WORDN 1 3)) ==>
        (~STABLE_LO (bsig I_last_O p')
                   (ti'+1,(@u'.NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti'+1)u')) /\          ~STABLE_LO
(bsig I_last_O p')
                    ((@u'. NTH_TIME_FALSE 0(bsig I_srdy_E e')(ti' + 1)u') + 1,
                     (@u'. NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti' + 1)u')) /\
         ~STABLE_LO (bsig I_last_O p')
                    ((@u'. NTH_TIME_FALSE 1(bsig I_srdy_E e')(ti' + 1)u') + 1,
                     (@u'. NTH_TIME_FALSE 2(bsig I_srdy_E e')(ti' + 1)u')) /\
         STABLE_HI (bsig I_last_O p')
                   (ti'+1,(@u'.NTH_TIME_FALSE 2(bsig I_srdy_E e')(ti'+1)u')) /\
         STABLE_LO (bsig I_last_O p')
                   ((@u'. NTH_TIME_FALSE 2(bsig I_srdy_E e')(ti' + 1)u')+1,
                    (@u'. NTH_TIME_FALSE 3(bsig I_srdy_E e')(ti' + 1)u')))"
   );;

let SETUP_TAC tm =
    REPEAT STRIP_TAC
    THEN IMP_RES_TAC ABS_SET_IMP_ABS
    THEN NRULE_ASSUM_TAC
           ("!pti t. PTAbs pti s e p t s' e' p'",
            ((SPECL [tm;"t:timeT"]) o (REWRITE_RULE [PTAbs])))
    THEN POP_ASSUM_LIST (MAP_EVERY (\thm. STRIP_ASSUME_TAC thm))
    THEN RES_TAC
    THEN RES_TAC
    THEN POP_ASSUM (\thm. ALL_TAC) %KEEP%
    THEN POP_ASSUM (\thm. ALL_TAC) %KEEP%
    THEN IMP_RES_TAC NTH_IBUS_TRANS_EXISTS
    THEN IMP_RES_TAC NTH_TIME_TRUE_X_IMP_X
    THEN POP_ASSUM (\thm. ALL_TAC) %KEEP%
    THEN RES_TAC
    THEN POP_ASSUM (\thm. ALL_TAC) %KEEP%
    THEN POP_ASSUM (\thm. ALL_TAC) %KEEP%;;

let lemma1 = TAC_PROOF
   (([], "! (x y :*) (f :*->**) . (x = y) ==> (f x = f y)"),
    REPEAT STRIP_TAC
    THEN ASM_REWRITE_TAC[]
   );;

let BS_WRITE = TAC_PROOF
   (([],
    "! (s :timeT->pt_state) (e :timeT->pt_env) (p :timeT->pt_out)
       (s' :timeC->pc_state) (e' :timeC->pc_env) (p' :timeC->pc_out) .
     PCSet_Correct s' e' p' ==>
       PTAbsSet s e p s' e' p' ==>
         PT_Exec PT_Write s e p t ==>
           PT_PreC PT_Write s e p t ==>
             (IB_BS_outO (PT_WriteOF (s t) (e t)) = IB_BS_outO (p t))"),
    SETUP_TAC "PT_Write"
    THEN IMP_RES_TAC (EXPAND_LET_RULE IB_BS_out_ISO)
    THEN IMP_RES_TAC (EXPAND_LET_RULE PB_BS_in_ISO)
    THEN ASM_REWRITE_TAC [PT_WriteOF_EXP;IB_BS_outO]
    THEN POP_ASSUM (\thm. ALL_TAC)
    THEN POP_ASSUM (\thm. ALL_TAC)
    THEN SUBGOAL_THEN
           "Standard_Assumps PT_Write s e p t s' e' p' tp' ti'" ASSUME_TAC
    THENL [
       % Subgoal 1: (New subgoal) %
       ASM_REWRITE_TAC [Standard_Assumps]
    ;
       % Subgoal 2: (Continue) %
       ASSUME_TAC
         (SPECL ["1";"SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0)"] MAXWORD)
       THEN IMP_RES_TAC (REWRITE_RULE [PRE_SUB1] LT_IMP_LE_PRE)
       THEN RULE_ASSUM_TAC REDUCE_RULE
       THEN ASSUME_TAC (SPEC "SND(L_ad_inE(e' (tp':timeC)))" SIZE_SUBARRAY_1)
       THEN IMP_RES_TAC LESS_EQ_3_CASES
       THENL [
          % Subgoal 2.1:  [ "VAL 1(SUBARRAY(SND(L_ad_inE(e' tp')))(1,0)) = 0" ] %
          IMP_RES_TAC
            (ISPECL ["VAL 1(SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0))";"0";
                     "WORDN 1"] lemma1)
          THEN IMP_RES_TAC WORDN_VAL_IDENT_1
          THEN ASM_REWRITE_ASSUM_TAC
                 ("WORDN 1(VAL 1(SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0))) =
                   WORDN 1 0",[])
          THEN IMP_RES_TAC I_LAST_FOR_BLOCK_SIZE_0'
          THEN ASM_REWRITE_TAC[]
       ;
          % Subgoal 2.2:  [ "VAL 1(SUBARRAY(SND(L_ad_inE(e' tp')))(1,0)) = 1" ] %
          IMP_RES_TAC
            (ISPECL ["VAL 1(SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0))";"1";
                     "WORDN 1"] lemma1)
          THEN IMP_RES_TAC WORDN_VAL_IDENT_1
          THEN ASM_REWRITE_ASSUM_TAC
                 ("WORDN 1(VAL 1(SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0))) =
                   WORDN 1 1",[])
          THEN IMP_RES_TAC I_LAST_FOR_BLOCK_SIZE_1'
          THEN ASM_REWRITE_TAC[]
       ;
          % Subgoal 2.3:  [ "VAL 1(SUBARRAY(SND(L_ad_inE(e' tp')))(1,0)) = 2" ] %
          IMP_RES_TAC
            (ISPECL ["VAL 1(SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0))";"2";
                     "WORDN 1"] lemma1)
          THEN IMP_RES_TAC WORDN_VAL_IDENT_1
```

```
            THEN ASM_REWRITE_ASSUM_TAC
                   ("WORDN 1(VAL 1(SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0))) =
                     WORDN 1 2",[])
            THEN IMP_RES_TAC I_LAST_FOR_BLOCK_SIZE_2'
            THEN ASM_REWRITE_TAC[]
         ;
            % Subgoal 2.4:  [ "VAL 1(SUBARRAY(SND(L_ad_inE(e' tp')))(1,0)) = 3" ] %
            IMP_RES_TAC
              (ISPECL ["VAL 1(SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0))";"3";
                       "WORDN 1"] lemma1)
            THEN IMP_RES_TAC WORDN_VAL_IDENT_1
            THEN ASM_REWRITE_ASSUM_TAC
                   ("WORDN 1(VAL 1(SUBARRAY(SND(L_ad_inE(e' (tp':timeC))))(1,0))) =
                     WORDN 1 3",[])
            THEN IMP_RES_TAC I_LAST_FOR_BLOCK_SIZE_3'
            THEN ASM_REWRITE_TAC[]
         ]
      ]
   );;
```

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** November, 1993	**3. REPORT TYPE AND DATES COVERED** Contractor Report

4. TITLE AND SUBTITLE Towards the Formal Verification of the Requirements and Design of a Processor Interface Unit--HOL Listings	**5. FUNDING NUMBERS** C NAS1-18586 WU 505-64-10-07
6. AUTHOR(S) David A. Fura, Phillip J. Windley*, and Gerald C. Cohen	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Boeing Defense & Space Group P.O. Box 3707, M/S 4C-70 Seattle, WA 98124-2207	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration NASA Langley Research Center Hampton, VA 23681-0001	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** NASA CR-191466

11. SUPPLEMENTARY NOTES

Langley Technical Monitor: Sally C. Johnson
Task 10 Report
*University of Idaho, Moscow, ID

12a. DISTRIBUTION / AVAILABILITY STATEMENT Unclassified - Unlimited Subject Category 62	**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(Maximum 200 words)***

This report contains the Higher-Order Logic (HOL) listings from the verification of the design and partial requirements for a Processor Interface Unit (PIU). The PIU is an interface chip performing memory interface, bus interface, and additional support services for a commercial microprocessor within a fault-tolerant computer system. General-purpose HOL theories and definitions that support the PIU verification as well as tactics used in the PIU proofs are presented. An informal description of the PIU verification effort is presented in a companion NASA contractor report entitled "Towards the Formal Verification of the Requirements and Design of a Processor Interface Unit."

14. SUBJECT TERMS Formal methods; Formal specification; Formal verification; Fault tolerance; Reliability; Specification	**15. NUMBER OF PAGES** 181
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**